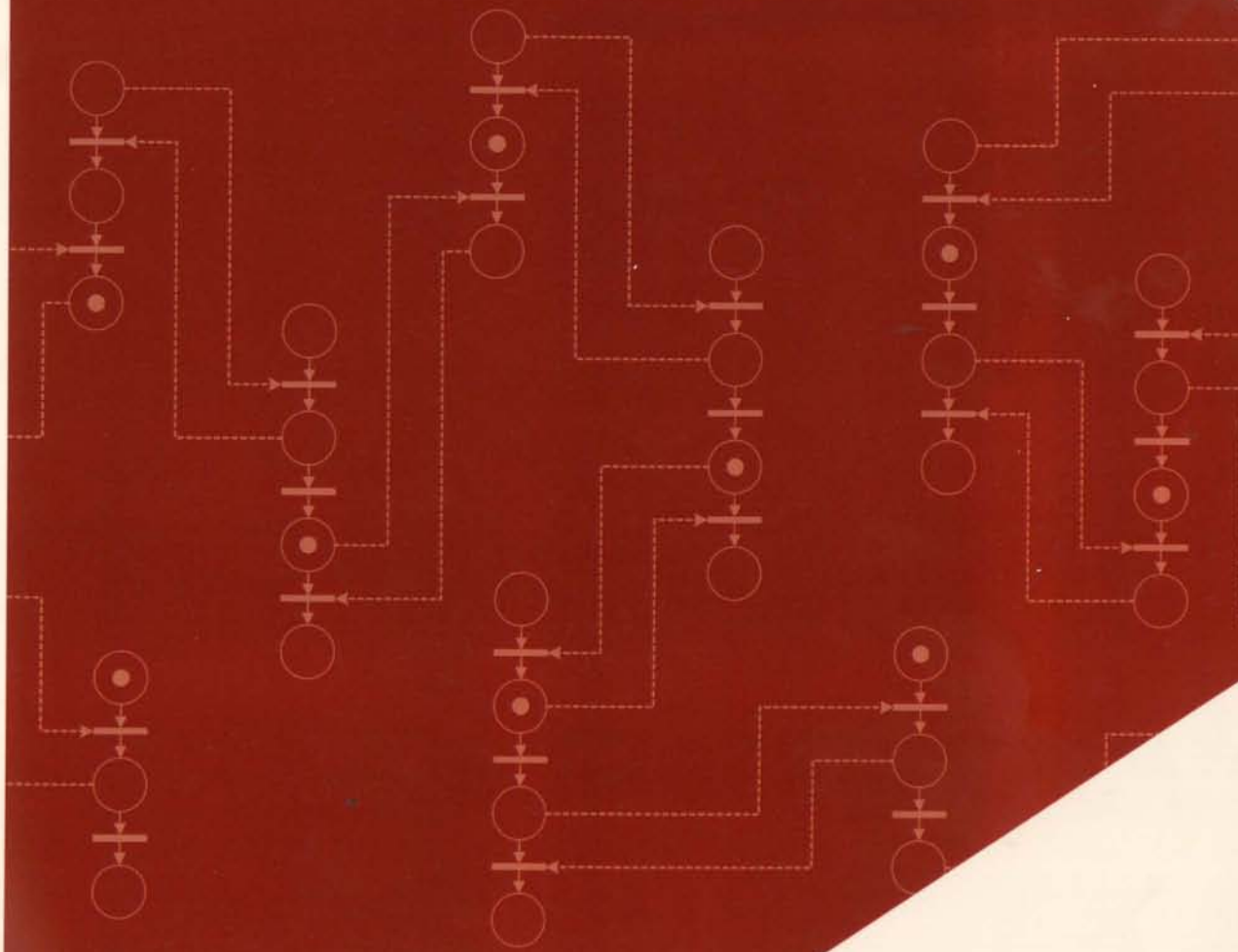
Diogo Manuel Ribeiro Ferreira

Workflow Management Systems Supporting the Engineering of Business Networks



Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto _ 2003

Workflow Management Systems Supporting the Engineering of Business Networks

DIOGO MANUEL RIBEIRO FERREIRA

Workflow Management Systems Supporting the Engineering of Business Networks

Dissertação apresentada à Faculdade de Engenharia da Universidade do Porto
para a obtenção do grau de Doutor em Engenharia Electrotécnica e de Computadores
realizada sob a orientação científica do Doutor João José da Silva e Cunha Pinto Ferreira,
Professor Associado da Faculdade de Engenharia da Universidade do Porto

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES
FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

2003

UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 79671
CDU 62.1.3 (043)
Data 19 / 3 / 2004

ISBN 972-9051-32-1

Cover design: CAETSU Publicidade S.A.

Printed in Portugal, July 2003

This work has been funded by:
Este trabalho foi financiado por:

Fundação para a Ciência e a Tecnologia (FCT)
http://www.fct.mct.pt

Abstract

The Internet has an unparalleled potential to reshape the way companies conduct their businesses. As a globally connected and widely accessible network infrastructure, the Internet allows an enterprise to integrate its business processes with those of its business partners. At the same time, it allows an enterprise to discover and establish business relationships with other business partners as well. Provided with a common, global network infrastructure, enterprises can associate with each other in order to become more competitive or to offer improved products or services. The result is the formation of business networks, which combine the competencies of several business partners.

This scenario requires enterprises to be able to integrate and coordinate business processes that extend beyond their borders. For this purpose, enterprises need a technological solution that provides them with flexibility regarding the kind of business processes they can establish with each other, and that provides them with control over those processes while respecting the autonomy of each business partner.

The ability to define and control the execution of business processes has always been the main focus of workflow management systems. However, workflow management systems have been typically employed within a single enterprise. More recently, some research projects have turned to the possibility of applying workflow management in inter-enterprise environments. This thesis is a study of the design issues and technological options that are relevant to the development of workflow management systems that address the integration of inter-enterprise business processes.

Based on a survey of existing workflow management systems, and on perspectives from the field of Enterprise Integration, this work establishes the fundamental result that every workflow management system should be developed as a combination of a workflow enactment service and an integration infrastructure. These two components have different requirements, and should be designed separately. A workflow management system that supports inter-enterprise business processes calls for an integration infrastructure that is suitable for inter-enterprise environments.

The thesis describes the main technological options for an inter-enterprise integration infrastructure by presenting a study of some of the most important business-to-business integration approaches. However, the experiences from two research projects suggest that only a completely decentralized infrastructure can succeed in an environment such as a business network, which comprises a set of autonomous business partners and therefore favors horizontal rather than hierarchical business structures.

On the other hand, the thesis identifies the common and desirable features for any workflow enactment service in general, and suggests that, instead of developing every new workflow management system from scratch, the development of workflow solutions could be based on a reusable core of workflow functionality which provides basic process modeling and execution capabilities.

The proposed solution to the integration of inter-enterprise business processes is a combination of a reusable workflow enactment service and a peer-to-peer integration infrastructure. The proposed workflow management system can support the whole life cycle of business relationships, from partner search to selection, contracting, fulfillment and evaluation - this process is referred to as the engineering of a business network. The result is also a generalized solution that can support both intra- and inter-enterprise business processes.

Résumé

L'Internet offre des possibilités intéressantes et inégalées de remodeler la manière que les entreprises conduisent leurs affaires. Comme une infrastructure globalement reliée et largement accessible, l'Internet permet aux entreprises d'intégrer leurs processus d'affaires avec ceux de leurs partenaires. En autre, elle permet aussi à une entreprise de découvrir et établir des rapports d'affaires avec d'autres partenaires. Avec une infrastructure de réseau commun, les entreprises peuvent s'associer entre elles afin de devenir plus concurrentielles ou afin d'offrir les produits et les services améliorés. Le résultat c'est la formation de réseaux d'affaires, qui combinent les compétences de plusieurs partenaires.

Ce scénario exige que les entreprises soient capables d'intégrer et coordonner des processus d'affaires qui se prolongent au delà de leurs frontières. À cette fin, les entreprises ont besoin d'une solution technologique qui leur fournit de la flexibilité concernant le genre de processus d'affaires qu'elles peuvent établir avec d'autres entreprises, et qui leur fournit le contrôle de ces processus-lá tout en respectant l'autonomie de chaque partenaire.

La capacité de définir et commander l'exécution des processus d'affaires a toujours été le principal objectif des systèmes de *workflow*. Cependant, les systèmes de workflow sont typiquement utilisés au sein d'une seule entreprise. Plus récemment, quelques projets de recherche se sont tournés vers la possibilité d'appliquer la gestion de workflow dans un environnement de plusieurs entreprises. Ce travail est une étude des issues de conception et des options technologiques qui sont remarquables pour le développement des systèmes de workflow qui permettent l'intégration des processus d'affaires entre des entreprises.

Basé sur un aperçu des systèmes de workflow existants, et sur des perspectives relatifs à l'intégration d'entreprise, ce travail aboutit à une conclusion fondamentale: n'importe quel système de workflow doit être développé comme une combinaison entre un service d'exécution de workflow et une infrastructure d'intégration. Un système de workflow qui soutient l'intégration de processus d'affaires entre des entreprises a besoin d'une infrastructure d'intégration convenable.

Ce travail décrit les principales options technologiques pour une infrastructure d'intégration entre des entreprises en présentant une étude de quelques des approches d'intégration *business-to-business* le plus important. Cependant, l'expérience de deux projets de recherche suggère que seulement une infrastructure complètement décentralisée est convenable à un réseau d'affaires, qui comporte un ensemble de partenaires autonomes et qui favorise donc plutôt les structures horizontales d'affaires que les structures hiérarchiques.

Dautre part, ce travail identifie les caractéristiques communes et souhaitables pour n'importe quel service d'exécution de workflow en général, et suggère que, au lieu de développer chaque nouveau système de workflow depuis le début, le développement de solutions de workflow pourrait être basé sur un noyau réutilisable de la fonctionnalité de workflow qui fournit tout de suite les possibilités de base pour modeler et exécuter des processus.

La solution proposée à l'intégration des processus d'affaires entre des entreprises est une combinaison entre un service réutilisable d'exécution de workflow et une infrastructure d'intégration du genre peer-to-peer. Le système de workflow proposé peut soutenir tout le cycle de vie des rapports d'affaires, dès la recherche de partenaires jusqu'aux moments de sélection, de contrat, d'operation et d'évaluation. Tout ce processus est désigné par "génie" d'un réseau d'affaires. Le résultat est aussi une solution généralisée qui peut soutenir des processus d'affaires dans et entre des entreprises.

Zusammenfassung

Das Internet hat ein unvergleichliches Potential, die Art und Weise umzugestalten, damit Unternehmen ihre Geschäfte leiten. Da das Internet eine global verbundene und zugängliche Netzinfrastruktur ist, erlaubt es den Unternehmen ihre Geschäftsprozesse mit denen ihrer Teilhaber zu integrieren. Außerdem läßt es ein Unternehmen gleichzeitig Geschäftsbeziehungen zu anderen Teilhabern zu finden und aufzubauen. Versehen mit einer verbreiteten Netzwerkinfrastruktur, können die Unternehmen mit einander verbunden werden. Das Ergebnis ist der Aufbau der Geschäftsnetze, die die Kompetenzen mehrerer Teilhaber verbinden.

Dieses Szenario verlangt von den Unternehmen Geschäftsprozesse zu integrieren und zu koordinieren, die sich über ihre Grenzen hinaus ausdehnen. Zu diesem Zweck benötigen Unternehmen eine technologische Lösung, die zum einen sie mit der Flexibilität hinsichtlich der Geschäftsprozesse versieht, die sie mit anderen Unternehmen herstellen können, und die zum anderen die Steuerung von Geschäftsprozessen ermöglicht, die die Autonomie jedes Teilhabers garantiert.

Die Kapazität die Geschäftsprozesse zu definieren und seine Durchführung zu steuern ist immer das Hauptziel der Workflow-Management-Systeme gewesen. Jedoch wurden die Workflow-Management-Systeme normalerweise innerhalb von Unternehmen angewandt. Kürzlich haben einige Forschungsprojekte sich mit der Möglichkeit beschäftigt, das Workflow-Management zwischen Unternehmen zu nützen. Es geht in diese Dissertation um die technologischen Optionen, die für die Entwicklung der Workflow-Management-Systeme relevant sind, und deshalb die Integration der Geschäftsprozesse zwischen Unternehmen ermöglicht.

Diese Dissertation gründet sich auf einer Übersicht der vorhandenen Workflow-Management-Systeme und auf Perspektiven im Bereich der Unternehmensintegration, und führt zu der Schlussforderung: dass nämlich jegliches Workflow-Management-System als Kombination eines Workflow Enactment Services und einer Integrationsinfrastruktur entwickelt werden sollte. Diese zwei Elemente haben unterschiedliche Anforderungen und sollten separat entwickelt werden.

Die Dissertation beschreibt die hauptsächlichen technologischen Optionen für die Infrastruktur der Integration zwischen den Unternehmen, durch eine Studie von einer der wichtigsten business-to-business Technologien. Jedoch die Erfahrung von zwei Forschungsprojekten zeigt dass nur eine Infrastruktur für das Umfeld des Geschäftsnetzes adäquat ist, die vollkommen dezentralisiert ist, weil ein Geschäftsnetz autonome Teilhaber einschließt und deshalb den Boden ebnet für horizontale Geschäftsstrukturen statt hierarchischen.

Andererseits kennzeichnet die Dissertation die allgemeinen und wünschenswerten Charakteristiken für jegliches Workflow Enactment Service im allgemeinen, und schlägt vor statt jedes neue Workflow-Management-System wieder von Grund auf zu entwickeln, dass ein Workflow-Management-System auf dem Boden eine wiederverwendbaren Funktionalität entwickelt werden kann. Diese wiederverwendbaren Funktionalität liefert schon die Grundkapazitäten für die Modellierung und Ausführung von Prozessen.

Die vorgeschlagene Lösung zur Integration der Geschäftsprozesse zwischen Unternehmen ist eine Kombination eines wiederverwendbaren Workflow Enactment Service und einer Integrationsinfrastruktur gegründet auf Peer-to-Peer. Das vorgeschlagene Workflow-Management-System kann die Geschäftsbeziehungen unterstützen, von der Suche nach Teilhabern bis zu deren Auswahl, Vertragsschliessung, Ausführung und Evaluation. Dieser Prozess ist gekennzeichnet durch den Aufbau eines Geschäftsnetzes. Das Ergebnis ist auch eine allgemeine Lösung, die die Geschäftsprozesse innerhalb und zwischen Unternehmen stützen kann.

Resumo

A Internet tem um enorme potencial para revolucionar o modo como as empresas conduzem os seus negócios. Por ser uma infra-estrutura interligada e acessível a nível global, a Internet permite às empresas integrar os seus processos de negócio com os dos seus parceiros de negócio. Por outro lado, a Internet também permite a cada empresa descobrir e estabelecer relações de negócio com outros parceiros de negócio. Com uma infra-estrutura de rede comum e global, as empresas podem associar-se entre si por forma a tornarem-se mais competitivas ou por forma a oferecerem melhores produtos ou serviços. O resultado é a formação de redes de negócio que reúnem as competências de vários parceiros de negócio.

Este cenário exige que as empresas sejam capazes de integrar e coordenar processos de negócio que se estendem para além das suas fronteiras. Para este propósito, as empresas necessitam de uma solução tecnológica que lhes ofereça flexibilidade relativamente aos processos de negócio que podem estabelecer com outras empresas e que providencie controlo sobre esses processos, salvaguardando ao mesmo tempo a autonomia de cada parceiro de negócio.

A capacidade de definir e controlar a execução de processos de negócio sempre foi o principal objectivo dos sistemas de gestão de fluxos de trabalho (sistemas de *workflow*). Contudo, os sistemas de gestão de fluxos de trabalho têm sido utilizados tipicamente no ambiente de uma única empresa. Recentemente, alguns projectos de investigação têm vindo a debruçar-se sobre a possibilidade de aplicar a gestão de fluxos de trabalho em ambientes inter-empresariais. Este trabalho apresenta um estudo das questões de concepção e opções tecnológicas que são relevantes para o desenvolvimento de sistemas de gestão de fluxos de trabalho que permitem a integração de processos de negócio inter-empresariais.

Baseada num estudo dos sistemas de gestão de fluxos de trabalho existentes e em perspectivas do ramo de Integração Empresarial, este trabalho chega a uma conclusão fundamental: qualquer sistema de gestão de fluxos de trabalho deve ser desenvolvido como uma combinação de, por um lado, um serviço de execução de fluxos de trabalho e, por outro, uma infra-estrutura de integração. Estas duas componentes têm requisitos diferentes e devem ser concebidas separadamente.

Este trabalho descreve as principais opções tecnológicas para uma infra-estrutura de integração inter-empresarial através do estudo de algumas das mais importantes abordagens de integração *business-to-business*. Porém, a experiência de dois projectos de investigação sugere que só uma infra-estrutura completamente descentralizada é adequada para o ambiente de uma rede de negócio, que abrange um conjunto de parceiros de negócio autónomos e que, por isso, favorece estruturas de negócio horizontais em vez de hierárquicas.

Por outro lado, este trabalho identifica as características comuns e desejadas para qualquer serviço de execução de fluxos de trabalho em geral e sugere que, em vez de desenvolver cada novo sistema de gestão de fluxos de trabalho desde o princípio, o desenvolvimento de soluções de gestão de fluxos de trabalho poderia partir de um núcleo de funcionalidade reutilizável que já ofereça as capacidades básicas de modelação e execução de processos.

A solução proposta para a integração de processos de negócio inter-empresariais é uma combinação de um serviço reutilizável de execução de fluxos de trabalho e de uma infra-estrutura de integração do tipo *peer-to-peer*. O sistema de gestão de fluxos de trabalho proposto consegue suportar todo o ciclo de vida das relações de negócio, desde a procura de parceiros até à sua selecção, contratação, operação e avaliação. Todo este processo é designado por engenharia da rede de negócio. O resultado é também uma solução generalizada de suporte a processos de negócio intra- e inter-empresariais.

Acknowledgements

Carrying out this work has been a long and often solitary endeavor. Yet, many people have directly or indirectly contributed to its successful completion.

I am especially indebted to my supervisor, Prof. João José Pinto Ferreira, for his friendship and support throughout this work. In addition, Prof. João José Pinto Ferreira has been instrumental in allowing me to participate and contribute to the research projects described in chapter 6 of this thesis.

I am also very grateful to Hugo Ferreira, my colleague and companion during the research projects we have participated in. Hugo has been the best teammate that one could possibly have.

I would like to thank several other people at INESC Porto. Luís Carneiro, the head of our unit, has made a consistent effort to ensure that I always had appropriate conditions to do my work. Paula Silva has provided important information on quality management issues and on ISO quality-related standards. Andreia Pereira has given valuable information and insights into the European Innovation Relay Centre (IRC) Network. César Toscano has supplied much of the bibliographic material for section 5.3 of this thesis, as well as useful comments on the technologies presented in chapter 5.

I would like to thank SMD Informática, now ParaRede, for having provided a fully working license of the OfficeWorks workflow management system for evaluation at INESC Porto, together with software manuals and permission to use some illustrations. I would like to thank Teodora Oliveira at EFACEC Sistemas de Informação S.A. for having provided technical information about Staffware's workflow management system, and Rehana Rafi at Staffware PLC for giving permission to use some illustrations.

I would like to thank the Martin-Luther-University of Halle-Wittenberg, in Germany, for having provided access to its library and facilities during my stay in November 2001. Some bibliographic material, notably the references in German, have been accessed at that library.

I would like to thank Prof. Ricardo Rabelo from the Universidade Federal de Santa Catarina (UFSC), Brazil, for having provided several pictures of the SC^2 software module, and Paulo Rodrigues at ParaRede for permission to reproduce illustrations from the SALSA software manuals. From the Swedish Institute of Computer Science (SICS), I would like to thank Daniel Gillblad for permission to use illustrations of software developed at SICS, as well as Anders Holst for help on technical issues concerning the preparation of this document. I would like to thank Manuela Ramos and Elisabeth Völpel for having translated the abstract to French and German, respectively.

Finally, I would like to acknowledge the constant support of my family and closest friends, whom I shall have the opportunity to thank personally.

Contents

Abstract 5

Résumé 7

Zusammenfassung 9

Resumo 11

Acknowledgements 13

1 Introduction 29
- 1.1 A new business environment . . . 29
- 1.2 Aiming at the business processes . . . 30
- 1.3 Outline of the thesis . . . 31

2 The Impact of the Internet on Business Practices 33
- 2.1 From networks to e-business . . . 33
  - 2.1.1 The ARPANET . . . 34
  - 2.1.2 The TCP/IP protocol . . . 34
  - 2.1.3 Towards a widespread infrastructure . . . 35
  - 2.1.4 The Internet . . . 36
  - 2.1.5 The World Wide Web . . . 37
  - 2.1.6 The WWW explosion . . . 38
- 2.2 Market trends and approaches . . . 38
  - 2.2.1 The start-ups versus the large companies . . . 39
  - 2.2.2 The evolution of B2C e-commerce . . . 40
    - 2.2.2.1 e-Payment . . . 40
    - 2.2.2.2 e-Banking and other financial e-services . . . 41
    - 2.2.2.3 The move to the wireless world . . . 42
  - 2.2.3 Characterizing the e-customer . . . 43
  - 2.2.4 The evolution of B2B e-commerce . . . 44
    - 2.2.4.1 Electronic data interchange . . . 44
    - 2.2.4.2 Internet technologies . . . 45
    - 2.2.4.3 E-marketplaces . . . 45
  - 2.2.5 Other forms of e-business . . . 46
    - 2.2.5.1 Public services and B2G . . . 46

2.2.5.2 B2E, e-learning and e-recruiting 47
2.3 Trends in Supply Chain Management 48
2.3.1 Increased intimacy with business partners 48
2.3.1.1 VMI 49
2.3.1.2 CPFR 49
2.3.1.3 Scan-based trading 49
2.3.1.4 Procure-to-pay transactions 50
2.3.1.5 Catalog management 50
2.3.1.6 Outsourcing 50
2.3.2 Automating and exposing business processes 50
2.3.2.1 Application-level automation 51
2.3.2.2 Business-level automation 52
2.3.3 The role of enterprise information systems 52
2.3.3.1 All-in-one versus specialized packages 54
2.3.3.2 Enterprise systems meet the Internet 55
2.4 Reengineering and re-organization 56
2.5 Concluding remarks 59

3 Workflow Management Systems **63**
3.1 Defining workflow 64
3.2 The Workflow Reference Model 66
3.2.1 Process definition tools 66
3.2.2 The workflow enactment service 67
3.2.3 Workflow client applications and invoked applications 68
3.2.4 Interoperability with other workflow enactment services 69
3.2.5 Administration and monitoring tools 69
3.3 Standardization efforts in workflow management 70
3.3.1 The Workflow Management Coalition (WfMC) 71
3.3.1.1 Interface 1 71
3.3.1.2 Interface 2 72
3.3.1.3 Interface 3 73
3.3.1.4 Interface 4 74
3.3.1.5 Interface 5 75
3.3.2 The Object Management Group (OMG) 77
3.3.3 The Internet Engineering Task Force (IETF) 81
3.4 Commercial applications 84
3.4.1 Staffware 87
3.4.1.1 Client/server technology 88
3.4.1.2 Staffware clients 89
3.4.1.3 The Staffware Workflow Server 89
3.4.1.4 Process modeling 91
3.4.1.5 Users, groups and roles 93
3.4.1.6 User interface for tasks 93
3.4.1.7 Invoking applications 94
3.4.1.8 Workflow client applications 95
3.4.1.9 Audit data 96

3.4.1.10 Compliance with standards . . . 96
3.4.2 OfficeWorks . . . 96
3.4.2.1 The correspondence log service . . . 97
3.4.2.2 The fax service . . . 97
3.4.2.3 The e-mail service . . . 98
3.4.2.4 The address book service . . . 99
3.4.2.5 The workflow service (Floway) . . . 100
3.5 Research applications . . . 103
3.5.1 System-centered workflow . . . 103
3.5.1.1 TriGSflow . . . 103
3.5.1.2 SWORDIES . . . 104
3.5.1.3 EvE . . . 105
3.5.1.4 APRICOTS . . . 105
3.5.1.5 ADEPT . . . 106
3.5.1.6 MENTOR . . . 107
3.5.1.7 WIDE . . . 109
3.5.1.8 WASA . . . 111
3.5.1.9 METEOR . . . 113
3.5.1.10 EXOTICA . . . 115
3.5.1.11 MOBILE . . . 115
3.5.2 Internet-mediated workflow . . . 116
3.5.2.1 Panta Rhei . . . 116
3.5.2.2 MAGI . . . 117
3.5.2.3 WW-flow . . . 117
3.5.2.4 MILOS . . . 118
3.5.3 Inter-enterprise workflow . . . 118
3.5.3.1 PRODNET II . . . 119
3.5.3.2 VEGA . . . 121
3.5.3.3 PLENT . . . 122
3.5.3.4 ACE-Flow . . . 123
3.5.3.5 WHALES . . . 124
3.5.3.6 WISE . . . 124
3.5.3.7 COSMOS . . . 127
3.5.3.8 CrossFlow . . . 129
3.6 Concluding remarks . . . 135

4 Perspectives from Enterprise Integration **139**
4.1 CIMOSA . . . 141
4.1.1 CIMOSA Modeling Framework . . . 141
4.1.2 CIMOSA Environments . . . 143
4.1.3 CIMOSA Integrating Infrastructure . . . 143
4.2 GERAM . . . 145
4.2.1 GERAM Framework components . . . 146
4.2.2 GERAM Entity life cycles . . . 148
4.2.3 GERAM Enterprise entities . . . 149
4.3 EMEIS . . . 150

4.3.1 Model Development Services (MDS) . . . 152
4.3.2 Model Execution Services (MXS) . . . 154
4.3.3 Shared and Base IT services . . . 157
4.4 Towards an inter-enterprise integration architecture . . . 157
4.4.1 Workflow Management and Enterprise Integration . . . 158
4.4.2 Business networking and Enterprise Integration . . . 160
4.4.3 The B2B trading life cycle . . . 161
4.4.4 The enterprise engineering life-cycle . . . 164
4.4.5 Matchmaking approaches . . . 165
4.5 Concluding remarks . . . 165

5 Infrastructures for B2B Interoperability **167**
5.1 Message oriented middleware (MOM) . . . 168
5.1.1 The Java Message Service (JMS) . . . 170
5.1.1.1 The message model . . . 171
5.1.1.2 The messaging facilities . . . 172
5.1.2 The CORBA Notification Service (CosNotify) . . . 175
5.1.2.1 The event channel . . . 175
5.1.2.2 Typed and untyped events . . . 176
5.1.2.3 Structured events . . . 178
5.1.2.4 QoS properties . . . 180
5.1.2.5 Event filtering . . . 182
5.1.2.6 Event translation . . . 183
5.2 Business-to-business frameworks . . . 184
5.2.1 OBI . . . 185
5.2.1.1 The OBI interaction model . . . 185
5.2.1.2 OBI documents and data objects . . . 186
5.2.1.3 Security measures . . . 187
5.2.2 cXML . . . 187
5.2.2.1 The PunchOut concept . . . 187
5.2.2.2 cXML messages . . . 188
5.2.2.3 Master agreements . . . 189
5.2.3 BizTalk . . . 189
5.2.3.1 Framework layers . . . 190
5.2.3.2 Message format . . . 190
5.2.3.3 Technology bindings . . . 191
5.2.4 bolero.net . . . 191
5.2.4.1 The Core Messaging Platform . . . 192
5.2.4.2 The Title Registry . . . 192
5.2.4.3 The settlement process . . . 192
5.2.4.4 boleroXML . . . 194
5.2.5 ebXML . . . 194
5.2.5.1 The ebXML registry . . . 194
5.2.5.2 Choreographies and collaboration activities . . . 195
5.2.5.3 Collaboration Protocol Profiles . . . 195
5.2.5.4 The ebXML Message Service . . . 197

5.2.5.5 Collaboration Protocol Agreements . . . 198
5.2.5.6 The ebXML functional view . . . 199
5.2.6 RosettaNet . . . 200
5.2.6.1 RosettaNet dictionaries . . . 200
5.2.6.2 Partner Interface Processes (PIPs) . . . 201
5.2.6.3 The RosettaNet Implementation Framework . . . 202
5.2.6.4 Trading Partner Agreements . . . 204
5.2.6.5 RosettaNet in action . . . 204
5.2.7 tpaML . . . 205
5.2.7.1 Structure of executable TPAs . . . 205
5.2.7.2 Service interfaces . . . 206
5.2.7.3 Code generation from executable TPAs . . . 207
5.2.8 eCo . . . 208
5.2.8.1 The eCo architecture layers . . . 208
5.2.8.2 Published interfaces . . . 208
5.2.8.3 eCo type registries . . . 211
5.2.8.4 Agent-based e-commerce . . . 212
5.3 Intelligent software agents . . . 212
5.3.1 Types of agents . . . 213
5.3.1.1 Mobile agents . . . 214
5.3.2 Multi-agent systems . . . 216
5.3.2.1 Agent communication languages (ACLs) . . . 216
5.3.2.2 Agent coordination . . . 218
5.3.3 Standardization efforts . . . 221
5.3.3.1 The FIPA specifications . . . 222
5.3.3.2 The MASIF facility . . . 222
5.3.4 Agent platforms . . . 223
5.3.5 Applications of software agents . . . 226
5.3.5.1 B2C e-commerce . . . 226
5.3.5.2 B2B e-commerce . . . 227
5.3.5.3 Manufacturing and Enterprise Integration . . . 228
5.3.5.4 Supply Chain Management and Virtual Enterprises . . . 231
5.3.6 Agents and workflow management . . . 234
5.3.6.1 Workflow management in agent-based systems . . . 234
5.3.6.2 Agents in workflow management systems . . . 235
5.3.6.3 Agents, workflow management, and the virtual enterprise . . . 238
5.4 Web services . . . 240
5.4.1 General architecture . . . 241
5.4.1.1 SOAP . . . 242
5.4.1.2 WSDL . . . 243
5.4.1.3 UDDI . . . 244
5.4.2 Additional specifications . . . 247
5.5 Peer-to-peer networking . . . 247
5.5.1 SETI@home . . . 248
5.5.2 Gnutella . . . 249
5.5.3 Freenet . . . 253

5.5.4 Jabber . . . 256
5.5.5 Project JXTA . . . 257
5.5.5.1 JXTA Concepts . . . 258
5.5.5.2 The Endpoint Routing Protocol . . . 260
5.5.5.3 The Peer Resolver Protocol . . . 263
5.5.5.4 The Rendezvous Protocol . . . 264
5.5.5.5 The Peer Discovery Protocol . . . 266
5.5.5.6 The Peer Information Protocol . . . 267
5.5.5.7 The Pipe Binding Protocol . . . 269
5.5.5.8 Standard services . . . 270
5.5.5.9 Community services . . . 272
5.5.5.10 Security in JXTA . . . 274
5.6 Concluding remarks . . . 274

6 Experiments and Findings **279**
6.1 The PRONEGI project . . . 279
6.1.1 The functional operations . . . 281
6.1.1.1 Activity models and functional operations . . . 282
6.1.1.2 3-Tier client/server architectures . . . 283
6.1.1.3 User interface . . . 284
6.1.2 The Workflow Backbone . . . 285
6.1.2.1 Publish-subscribe communication . . . 285
6.1.2.2 The use of Contexts . . . 287
6.1.2.3 Support for functional operations . . . 288
6.1.2.4 System design . . . 291
6.1.2.5 Dynamic behavior . . . 293
6.1.2.6 System implementation . . . 295
6.1.3 The workflow enactment service (XFlow) . . . 296
6.1.3.1 Process modeling . . . 297
6.1.3.2 Activity parameters . . . 299
6.1.3.3 Process triggering . . . 299
6.1.3.4 User configuration . . . 300
6.1.3.5 The workflow client . . . 300
6.1.3.6 Activity assignment . . . 302
6.1.3.7 System implementation . . . 303
6.2 The DAMASCOS project . . . 305
6.2.1 The Business Functions . . . 307
6.2.1.1 SALSA, IDLS, and IPO . . . 308
6.2.1.2 D_3S_2 . . . 310
6.2.1.3 SC^2 . . . 311
6.2.2 The Workflow Backbone . . . 312
6.2.2.1 The underlying messaging platform . . . 315
6.2.2.2 Users, user groups and message filtering . . . 316
6.2.2.3 CORBA services and interfaces . . . 318
6.2.2.4 The COM interface layer . . . 320
6.2.2.5 The Workflow Facility . . . 322

6.2.2.6 Process modeling . . . 323
6.2.2.7 Process execution . . . 324
6.2.2.8 Implementation and testing . . . 327
6.2.2.9 Latest improvements . . . 328
6.2.2.10 Security and federation . . . 330
6.3 Findings . . . 332
6.3.1 Resource allocation . . . 334
6.3.2 Model granularity . . . 336
6.3.3 Activity transitions . . . 337
6.3.4 Multi-task activities and ad-hoc processes . . . 339
6.3.5 Observer pattern . . . 340
6.3.6 Integration infrastructure . . . 341
6.3.7 Electronic contracts and network-level processes . . . 342
6.3.8 Security . . . 343
6.4 Concluding remarks . . . 345

7 Proposed Solution **349**
7.1 Process modeling and execution services . . . 350
7.1.1 Reusable workflow enactment services . . . 351
7.1.2 Process modeling and workflow analysis . . . 353
7.1.2.1 The soundness property . . . 355
7.1.2.2 State-based vs. event-based modeling . . . 357
7.1.2.3 Colored Petri nets and process instances . . . 359
7.1.2.4 Stochastic Petri net models . . . 360
7.1.3 Interaction with resources . . . 361
7.1.3.1 Activities and actions . . . 362
7.1.3.2 Resource invocation . . . 362
7.1.3.3 Activity instances . . . 365
7.1.3.4 Activity input and output data . . . 365
7.1.4 The Workflow Kernel . . . 365
7.1.4.1 The IFeatures base interface . . . 366
7.1.4.2 Object-specific interfaces . . . 368
7.1.4.3 The IEnumerator base interface . . . 370
7.1.4.4 The INotifySink callback interface . . . 372
7.1.4.5 The event notification method . . . 374
7.1.4.6 Event sources and event sinks . . . 375
7.1.4.7 Handling properties and documents . . . 378
7.1.5 Implementing the Workflow Kernel . . . 381
7.1.5.1 Choosing the implementation technology . . . 381
7.1.5.2 The Workflow Kernel modules . . . 385
7.1.5.3 Aggregation of Enumerator objects . . . 386
7.1.5.4 Internal notification of events . . . 388
7.1.5.5 The use of Connection Points . . . 389
7.1.6 Reusing the Workflow Kernel . . . 390
7.1.6.1 Potential use in PRONEGI . . . 391
7.1.6.2 Potential use in DAMASCOS . . . 393

7.1.6.3 Modeling and monitoring tools . . . 394
7.1.6.4 Building workflow management systems . . . 395
7.2 Integration infrastructure and interaction services . . . 397
7.2.1 Building a P2P integration infrastructure . . . 398
7.2.1.1 Multicast and unicast services . . . 398
7.2.1.2 Main objects and interfaces . . . 400
7.2.1.3 Service message structure . . . 403
7.2.2 Integration with the Workflow Kernel . . . 405
7.2.2.1 The PeerSvcs module . . . 405
7.2.2.2 The IServiceListener interface . . . 407
7.2.2.3 Workflow actions and service messages . . . 410
7.2.2.4 Service messages and JXTA messages . . . 412
7.2.2.5 Pre-defined vs. customized properties . . . 412
7.2.2.6 The QueryID property . . . 415
7.3 Proposed integration architecture . . . 416
7.3.1 Service client/provider roles . . . 418
7.3.2 External support services . . . 418
7.3.3 Internal support services . . . 419
7.3.4 The engineering process . . . 420
7.3.5 Business networking . . . 437
7.4 Concluding remarks . . . 439

8 Conclusion **441**
8.1 Summary of main contributions . . . 441
8.2 Open issues and future work . . . 444
8.3 Closing statement . . . 445

References **447**

A The European Innovation Relay Centre Network **471**

List of Figures

2.1 Evolution of the enterprise towards a network of competencies 61

3.1 Evolution of office automation systems 64
3.2 The framework for workflow management 65
3.3 The Workflow Reference Model components and interfaces 67
3.4 Example state transitions for a process instance 68
3.5 Interoperability modes between workflow engines 70
3.6 Structure of a WPDL process definition file 72
3.7 The principle for the WfMC's interface 4 74
3.8 Structure of a CWAD data record 76
3.9 Class diagram for the OMG Workflow Management Facility 79
3.10 State transitions for a WfExecutionObject 80
3.11 Example of a SWAP request 83
3.12 Survey of workflow vendors and products (June 2002) 86
3.13 Staffware client/server architecture 88
3.14 Types of Staffware client applications 89
3.15 The Staffware Workflow Server software architecture 90
3.16 The Staffware physical system architecture 91
3.17 Manual users versus Staffware Brokers 95
3.18 Reading an OfficeWorks circular e-mail message 98
3.19 Front-end provided by Floway to edit flows 100
3.20 Front-end provided by Floway for defining a stage 101
3.21 Front-end provided by Floway for users to manage stages 102
3.22 Activity sequence according to ECA rules 104
3.23 Activity sequence according to ECAA rules 104
3.24 Architecture for the MENTOR project 108
3.25 Architecture for the WIDE project 109
3.26 Architecture for the WASA project 112
3.27 Nested graphs for process modeling in WASA 113
3.28 Evolution towards the METEOR architecture 114
3.29 Architecture for the PRODNET II project 120
3.30 Key concepts in WISE 125
3.31 Architecture for the COSMOS project 128
3.32 The contract as the means to configure internal resources in CrossFlow 130
3.33 Contract templates and the matchmaking facility in CrossFlow 131

3.34 Components of the CRAFT integration facilitator in CrossFlow 134
3.35 Lack of products and standards in inter-enterprise workflow 136

4.1 Levels of enterprise integration 140
4.2 CIMOSA Modeling Framework 142
4.3 CIMOSA Generic Building Blocks 143
4.4 CIMOSA life cycles 144
4.5 The CIMOSA Integrating Infrastructure 144
4.6 GERAM framework components 147
4.7 GERAM life cycle phases 149
4.8 Relationships between life cycles of GERAM entity types 150
4.9 Enterprise integration with and without executable models 152
4.10 EMEIS phases of model development 153
4.11 EMEIS phases of model execution 155
4.12 Workflow Management vs. GERAM and EMEIS 159
4.13 Relationships between enterprise life cycles within a business network 161
4.14 Relation between enterprise and business network life cycles 163
4.15 The enterprise life history 164

5.1 Integration technologies in the framework of EMEIS 168
5.2 Message passing (a), message queuing (b) and publish/subscribe (c) 169
5.3 Overview of MOM system vendors and products (August 2002) 170
5.4 Behavior of JMS clients 171
5.5 Simplified diagram of JMS interfaces 173
5.6 Some providers of CosNotify implementations (August 2002) 175
5.7 Push (a) and pull (b) models in the CORBA Event Service 176
5.8 Suppliers and consumers operating in push and pull models 176
5.9 The proxy supplier and proxy consumer interfaces 177
5.10 Interfaces specified by the CORBA Notification Service 178
5.11 Structured event in the CORBA Notification Service 179
5.12 Applicability of QoS properties at various levels 181
5.13 The OBI business-to-business interaction model 185
5.14 EDI (a) and XML (b) OBI data structures 186
5.15 Message sequence for a PunchOut session 188
5.16 (a) Request, (b) response, and (c) one-way cXML messages 189
5.17 The BizTalk framework layers 190
5.18 The BizTalk message format 191
5.19 Participants in the settlement process 193
5.20 Elements of the ebXML Collaboration Protocol Profile 196
5.21 Main components of the ebXML Message Service 198
5.22 Elements of the ebXML Collaboration Protocol Agreement 199
5.23 Simplified ebXML functional view 200
5.24 Typical PIP definition 202
5.25 The RosettaNet Business Message 203
5.26 Structure of tpaML documents 206
5.27 The eCo Architecture layers 209

5.28 The published interface for each eCo architecture layer 210
5.29 Response to the BusinessGetProperties query 210
5.30 Common queries for eCo type registries 212
5.31 KQML performatives between a sender (S) and a receiver (R) 217
5.32 FIPA ACL message structure 218
5.33 Elements of the STL++ coordination language 220
5.34 A trading system represented with STL++ 221
5.35 Areas addressed by FIPA standards 222
5.36 Some of the currently available agent platforms 224
5.37 Layers of the FIPA-OS architecture 225
5.38 Message structures for an agent-based electronic marketplace 233
5.39 Dynamic workflow service provisioning 235
5.40 Main players in the Web Services architecture 242
5.41 Layers of the Web Services architecture 242
5.42 General structure for SOAP messages 243
5.43 General structure for a WSDL service description 244
5.44 Relationships between the main UDDI data structures 245
5.45 Inquiry (a) and publishing (b) operations supported by Microsoft UBR 246
5.46 High-level architecture for the SETI@home project 249
5.47 The five descriptors for the Gnutella protocol 251
5.48 Descriptor routing in Gnutella 252
5.49 Keyword-signed keys (a) and signed-subspace keys (b) in Freenet 254
5.50 An example of a request sequence in Freenet 255
5.51 XML streams and chunks in Jabber 257
5.52 The JXTA software architecture 261
5.53 Message sequence in the JXTA Endpoint Routing Protocol 262
5.54 Structure of Endpoint Routing Protocol messages 263
5.55 Message sequence in the JXTA Peer Resolver Protocol 264
5.56 Structure of Peer Resolver Protocol messages 264
5.57 Message sequence in the JXTA Rendezvous Protocol 265
5.58 Structure of Rendezvous Protocol messages 266
5.59 Structure of Peer Discovery Protocol messages 267
5.60 Structure of Peer Information Protocol messages 268
5.61 Structure of pipe advertisements and Pipe Binding Protocol messages 270
5.62 Module advertisements 273
5.63 Technological options for B2B interoperability 275
5.64 Conceptual framework for different technological options 276

6.1 Sample parts manufactured by SONAFI 280
6.2 The PRONEGI system approach 281
6.3 Simplified TNC (a) and TR/D (b) processes using ARIS eEPC notation 283
6.4 User interface for a functional operation (Customer Claim) 284
6.5 The PRONEGI workflow architecture 286
6.6 The observer pattern 286
6.7 Publish-subscribe communication using Contexts 288
6.8 Procedure for determining whom to dispatch a Subject 288

6.9 The FuncOpsManager and the Workflow Backbone 289
6.10 The Functional Operations Manager user interface 290
6.11 Class diagram for the PRONEGI Workflow Backbone 292
6.12 Dispatch sequence across the Workflow Backbone 295
6.13 User interface for the Workflow Backbone 296
6.14 The TR/D process expressed as workflow process 297
6.15 Defining the TR/D process in the XFlow application 298
6.16 Activity input and output parameters 299
6.17 Triggering a process manually (a) and automatically (b) 300
6.18 Configuring system users in the XFlow application 301
6.19 Workflow clients in the PRONEGI system 301
6.20 Activity assignment protocol in PRONEGI 303
6.21 Activity rejection (a) and failure (b) 304
6.22 Decoupling MFC code and CORBA code 305
6.23 Retrieving product information using the SALSA module 309
6.24 Adding customer orders with the SALSA module 310
6.25 Client application used to test the D_3S_2 module 312
6.26 The SC^2 module 313
6.27 The DAMASCOS architecture 314
6.28 The P/S matrix 314
6.29 Specifying a document exchange sequence as a workflow process 315
6.30 WfBB users and CosNotify objects 317
6.31 Message filtering within the WfBB messaging platform 318
6.32 Main access interfaces to the WfBB Administration Service 319
6.33 Relationships between COM and CORBA objects 321
6.34 Editing WPDL files graphically with the ProcessEditor tool 324
6.35 Setting process triggering conditions with the ProcessEditor tool 325
6.36 Run-time behavior of the WfFacility and the WfBB 326
6.37 Geographical testing scenario for the DAMASCOS Suite 329
6.38 The challenge of federating several WfBB instances 331
6.39 A gateway application between WfBB instances 331
6.40 Activity assignment in PRONEGI (a) and DAMASCOS (b) 335
6.41 The process model and the resource allocation model 336
6.42 Intra-enterprise (a) and inter-enterprise (b) activity assignment 337
6.43 Nesting processes belonging to different environments 338
6.44 Conditional transitions based on boolean expressions (a) and events (b) 338
6.45 Example of a multi-task activity 339
6.46 The observer pattern in different scenarios 341
6.47 Network-level and enterprise-level processes 344

7.1 The proposed integration approach 350
7.2 Application integration using the Drala Workflow Engine 352
7.3 Architecture for the wftk toolkit 352
7.4 The WorkMovr layered API 353
7.5 Example of a sound WF-net 356
7.6 Deadlock (a), livelock (b) and branching mismatch (c) inconsistencies 356

7.7 Compositionality of WF-nets . . . 357
7.8 Event-based modeling . . . 358
7.9 Special cases in event-based modeling . . . 358
7.10 Equivalent representations of a producer-consumer system . . . 360
7.11 Dividing activities (a) into actions (b) . . . 363
7.12 Actions as the resource invocation layer . . . 364
7.13 Push and pull models in resource invocation . . . 364
7.14 Input and output properties (P) and documents (D) . . . 366
7.15 Simplified class diagram for the Workflow Kernel . . . 367
7.16 The IFeatures base interface and other Workflow Kernel interfaces . . . 368
7.17 The IEnumerator base interface and other container interfaces . . . 371
7.18 Relationship between container and object-specific interfaces . . . 372
7.19 Internal event notification with INotifySink . . . 373
7.20 External event notification with INotifySink . . . 374
7.21 The obj_type (a) and ntf_type (b) enumeration values . . . 376
7.22 Relationship between obj_type and ntf_type values . . . 376
7.23 Flow of notification events across the Workflow Kernel . . . 378
7.24 Process execution as a reaction to event notification . . . 379
7.25 The COM connection points architecture . . . 383
7.26 Reusing COM objects by aggregation . . . 384
7.27 Security in Windows NT/2000 (a) and Kerberos (b) . . . 385
7.28 Aggregated objects within a Place object . . . 387
7.29 Internal event notification to the outer object . . . 389
7.30 The event flow mechanism with COM connection points . . . 391
7.31 Modeling the TR/D process with the Workflow Kernel . . . 392
7.32 A new WfFacility based on the Workflow Kernel . . . 393
7.33 The WfKMonitor and the WfKEditor tool components . . . 396
7.34 Proposed architecture for workflow management systems . . . 397
7.35 Multicast (a) and unicast (b) services within a peer group . . . 399
7.36 Main classes in the integration infrastructure . . . 401
7.37 Multicast (a) and unicast (b) service messages . . . 404
7.38 Relationships between COM and CORBA interfaces . . . 406
7.39 Subscribing to messages via the PeerSvcs module . . . 408
7.40 Message notification up to the Workflow Kernel . . . 409
7.41 Modeling request/reply behavior with the Workflow Kernel . . . 411
7.42 Properties and documents across software layers . . . 413
7.43 Pre-defined properties for Actions and Events . . . 414
7.44 Pre-defined properties in Actions and Events . . . 415
7.45 Integration architecture for inter-enterprise business processes . . . 417
7.46 Linking inter-enterprise processes to intra-enterprise processes . . . 420
7.47 Workflow client application . . . 422
7.48 The buyer-side engineering process . . . 425
7.49 The buyer-side engineering process (continued) . . . 426
7.50 The buyer-side engineering process (continued) . . . 427
7.51 The buyer-side engineering process (continued) . . . 428
7.52 The buyer-side engineering process (continued) . . . 429

7.53 The buyer-side engineering process (continued) 430
7.54 The supplier-side engineering process 431
7.55 The supplier-side engineering process (continued) 432
7.56 The supplier-side engineering process (continued) 433
7.57 The supplier-side engineering process (continued) 434
7.58 The supplier-side engineering process (continued) 435
7.59 The supplier-side engineering process (continued) 436
7.60 Business relationships within a virtual organization 437
7.61 Business relationships within a business network 438
7.62 Proposed integration architecture for business networks 440

A.1 Structure of the European IRC Network 472
A.2 Matchmaking in the European IRC Network 473
A.3 Structure of technology profiles 474

Chapter 1

Introduction

We live in an increasingly connected world, and one of the major factors that has contributed to this scenario was the advent of the Internet. It is commonly said that the Internet has turned the world into a global village. The Internet is having a strong impact on our daily lives and on the way companies conduct business. As a medium, the Internet allows customers to reach companies and companies to reach customers; put simply, it brings them together. But the Internet also brings companies closer to other companies: it brings companies closer to their business partners; it also brings companies closer to their competitors. Therefore, the Internet fosters a new business environment: one in which information can be exchanged or disseminated quickly, one in which companies can implement closer relationships with their business partners, one in which companies can meet and conduct business with new business partners. The Internet is the fundamental infrastructure that will allow business partners to connect with each other.

1.1 A new business environment

Enterprises have always been struggling to produce with lower costs, with the required quality, and within a shorter time [Kimball and Kimball Jr., 1947]. This trend, which is so often invoked in present times, actually precedes the Internet and Information Technology (IT) in general. In fact, ever since the Industrial Revolution, progress has developed along three lines, namely labor-saving machinery, time-saving machinery and machinery for communicating information [Kimball and Kimball Jr., 1947]. If one looks at Information Technology, then these are precisely the benefits that it brings: it supports workers while they perform tasks, it enables workers to do their work faster, and it enhances the exchange of information between resources.

But if one is to delve deeper into the benefits of Information Technology, one finds that IT brings an additional kind of benefit, which can be referred to as *organizational enhancement* [Eason, 1988]. Information Technology fosters new forms of integration, both within and across enterprise borders; its aim is to provide the right information at the right place at the right time under the right format. Within the enterprise, Information Technology can fragment major functions into a network of cooperative, autonomous resources, decentralizing decision-making and flattening hierarchical structures [Vernadat, 1996]. Between enterprises, Information Technology can synchronize and coordinate the business processes running at different, autonomous organizations, as well as improve the ability of an enterprise to find and engage in business with any potential business partner.

The rapid advances in Information Technology are set to reshape the way enterprises conduct their own businesses. On one hand, enterprises do not have to perform all the work themselves; in an increasingly connected world, it is perhaps easier to find business partners that can perform some of the work better, faster and cheaper. On the other hand, enterprises do not have to restrict themselves to the products or services they can manufacture or render; in an increasingly connected world, enterprises can associate with each other in order to provide products or services that none of them would be able to provide alone. In essence, Information Technology allows enterprises to establish closer relationships with each other, creating new business structures. The act of developing business relationships supported by Information Technology is called *business networking* [Österle et al., 2001].

1.2 Aiming at the business processes

Typically, business processes within an enterprise are supported by monolithic, database-centered information systems that store all of the relevant information and are able to present it to the human user in several ways, whenever the user has some task to perform. But what these systems usually lack is precisely a way to explicitly define tasks, to define how they should be sequentially executed, and to enforce their execution according to a given process logic. These issues are addressed by another kind of systems called *workflow management systems*.

Business networking requires appropriate support tools in order to manage work that flows across enterprises borders. To implement a business relationship, enterprises must be able to somehow link their business processes. An enterprise could do this by having a worker send a fax to another enterprise whenever some operation is completed, or by writing a small program that automatically connects and sends the data to a remote information system at another enterprise. Regardless of how an enterprise actually does it, the fact is that the enterprise must have some way to coordinate its own internal activities with the activities performed by external business partners. Business networking requires enterprises to be able to integrate and coordinate business processes. This is a job for workflow management systems.

However, workflow management systems have been used mainly to support business processes in intra-enterprise environments. This work delves into the study and development of workflow management systems that can support the integration of business processes in inter-enterprise environments. It is a study of the main characteristics of existing workflow management systems, and of the design issues and technological options that are relevant to the development of workflow management systems that support business networking. The experience gained from two research projects, which have required the development of two workflow management systems, has led to a number of conclusions which would eventually shape the proposed solution to the integration of inter-enterprise business processes.

But the work presented in this thesis does not focus exclusively on how enterprises can integrate business processes with each other. It also focuses on how enterprises can discover each other and develop business relationships with each other, all of this supported by workflow management. Based on key ideas from enterprise integration architectures, this work distinguishes between operational processes and engineering processes. Operational processes are those take place when enterprises interact with each other in order to provide a product or service to an end-customer. Engineering processes are those that define how an enterprise searches for business partners and develops business relationships with them. The goal is to come up a workflow management system that can support both

operational and engineering processes, and to devise a single, generalized approach to the modeling and integration of intra- and inter-enterprise business processes.

1.3 Outline of the thesis

Chapter 2 begins by describing the birth of the Internet and how it has evolved into a business platform, having an impact on the relationships between companies and end customers, between companies and their business partners, and even between governments and the general population. Focusing on business-to-business relationships, chapter 2 then shows how the Internet and its associated technologies have encouraged new trends in supply chain management by allowing business partners to become more closely integrated. The possibilities of the Internet and its associated technologies have motivated significant improvements in information systems, providing companies with more control over its own operations and with the ability to exchange information in real-time with all of its business partners. Taking advantage of these possibilities, some companies have completely reshaped not only their business processes but also their organizational structure, turning themselves into virtual organizations.

The general trend which chapter 2 attempts to demonstrate is that, given the networking possibilities provided by the Internet and the fact that it creates a global marketplace that puts even more pressure on companies, each enterprise will tend to focus on their core competencies and will seek business partners with the remaining competencies in order to carry out its business purposes. Rather than slowly developing fixed channels between producers and consumers, as in the traditional supply chain, the enterprise fragments itself into a network of competencies where business processes must be quickly configured and tightly integrated across an arbitrary set of business partners. This kind of business structure, henceforth called a *business network*, requires a flexible and generic workflow solution that can support the life cycle of such business relationships.

Chapter 3 focuses on workflow management systems in general. It defines workflow and describes the standard Workflow Reference Model, which reflects the currently accepted view on workflow management. The Workflow Reference Model has been the rationale for several standards issued by the Workflow Management Coalition (WfMC). Besides the WfMC, other organizations have also developed workflow standards, namely the Object Management Group (OMG) and the Internet Engineering Task Force (IETF). After briefly presenting the standards, chapter 3 describes two software products that are thought to be representative of the commercial workflow management systems available today. The remaining of chapter 3 is devoted to the discussion of research projects in the area of workflow management systems. The discussion covers several research prototypes, from early systems, which were strongly related to database technology, to more recent projects, which have applied workflow management in inter-enterprise environments.

Chapter 4 provides a more insightful perspective of what *integration* is about, and suggests that, in spite of having its origin in office automation systems, workflow management systems can actually be regarded as a tool for enterprise integration. By means of process execution capabilities, workflow management systems can automate the *operational life cycle*, i.e., the sequence of phases during which enterprises interact and operate with each other, in the form of business-to-business trading relationships. But by means of process modeling capabilities, workflow management systems can also support the *engineering life cycle*, i.e., the design and specification of how each enterprise will allow those business-to-business relationships to develop. Chapter 4 also shows that enterprise integration is about developing enterprise models and executing them over an integration infrastructure. Workflow

management systems do provide modeling and execution capabilities, but they often lack a clear distinction between these capabilities and the integration services that only an appropriate integration infrastructure can provide.

Chapter 5 describes the main technological options that are currently available to develop an integration infrastructure for an inter-enterprise environment. Altogether, five major technologies are presented: message-oriented middleware, business-to-business frameworks, agent-based systems, Web services, and peer-to-peer networking which is thought to have the potential to become an equally important technology. Chapter 5 shows that each of these technologies leads to a different kind of infrastructure, and that they should not be regarded as being necessarily alternative options to each other. Chapter 5 also suggests that any of these technologies can be used in combination with workflow management, but that each of them has its own issues and requirements. From these technologies, message-oriented middleware seems to be the most straightforward option to deal with the basic need for message exchange during workflow execution, although the integration of business processes in an inter-enterprise environment will require a more decentralized infrastructure.

Chapter 6 describes the design and implementation of two workflow management systems that have been developed within the scope of two research projects. In PRONEGI, the purpose of developing a workflow management system was to coordinate and support the execution of Total Quality Management (TQM) procedures within a particular enterprise. In DAMASCOS, a European project, the purpose of developing a workflow management system was to integrate a set of software modules residing at the same or at different enterprises. The two projects have led to similar solutions: both systems comprise a messaging infrastructure which connects several applications, one of them being a workflow enactment service which controls the interaction between the remaining applications. However, one of the main findings was that the integration infrastructure, in both cases, was too centralized for an inter-enterprise environment. Facing the fact that decentralizing either solution was virtually impossible, it was realized that the desired workflow management system would have to be developed according to a radically different architecture.

Chapter 7 presents the proposed solution to the challenge of developing a workflow management system that supports the engineering of business networks by supporting the enterprise engineering life cycle and the business-to-business trading life cycle. The proposed workflow management system comprises a workflow enactment service and an integration infrastructure, but both of these components are entirely new. From the study of several workflow management systems and from the experience of the two research projects, it was possible to identify a set of basic features that every workflow enactment service should have. These features have been implemented as a reusable and extensible workflow enactment service, which can be used as the central core in other workflow management systems. On the other hand, the integration infrastructure has been developed as a completely decentralized platform, which allows an enterprise to discover and interact with other enterprises by means of a set of basic, common services. According to the proposed solution, each enterprise becomes a peer in a network of workflow-enabled nodes.

Chapter 8 summarizes the main contributions of this work and highlights some issues that deserve further investigation.

Chapter 2

The Impact of the Internet on Business Practices

From the beginning of the ARPANET to the latest forms of e-business, the purpose of this chapter is to show how the Internet promotes new business possibilities that will make enterprises reengineer their systems, and even reorganize themselves. The main driving factor is the fact that the Internet is a globally connected and widely accessible network infrastructure, and therefore provides unprecedented access to customers and potential business partners, as well as to competitors. As enterprises strive to become more competitive, they will necessary adopt a strategy based on dividing competencies among a set of business partners, just like, in the past, industry has improved its productive powers by dividing labor among a set of workers [Smith, 1776]. This chapter attempts to describe the evolution towards business networks as a sequence of steps that, having started from the rise of the Internet (section 2.1), will eventually lead to a business landscape dominated by dynamic business relationships. The chapter discusses how the Internet has first affected the market (section 2.2), then the relationships between enterprises (section 2.3), and afterwards the structure of enterprises themselves (section 2.4).

2.1 From networks to e-business

If the developments and breakthroughs that have fostered the rise of the Internet were to be tracked to a single event, that event would probably be the remote connection of two computers between UCLA (University of California Los Angeles) and SRI (Stanford Research Institute) on 2nd September 1969 [Zakon, 2001]. Leonard Kleinrock, still a professor at UCLA nowadays, describes the login attempt onto the remote SRI computer with the following words [Gromov, 2000]:

> "We set up a telephone connection between us and the guys at SRI (...) We typed the L and we asked on the phone: - «Do you see the L?» - «Yes, we see the L» came the response. - We typed the O, and we asked, «Do you see the O?» - «Yes, we see the O.» - Then we typed the G, and the system crashed yet a revolution had begun..."

Shortly after UCLA and SRI had been successfully connected, UCSB (Santa Barbara) was connected to both UCLA and SRI, and the University of Utah in Salt Lake City was connected to SRI, making four network nodes by the end of 1969. These were the first nodes of the Internet.

2.1.1 The ARPANET

In reaction to the USSR launch of Sputnik in October 1957, the US government saw the need for an Advanced Research Projects Agency (ARPA) which would eventually build the first US satellite in 18 months [Gromov, 2000]. ARPA was subsequently focusing on communications technology and in improving the military use of computer technology. J. R. Licklider from MIT (Massachusetts Institute of Technology), who coined the term *Galactic Network* in August 1962 envisioning a globally interconnected set of computers, would become the head of research for ARPA in October the same year [Leiner et al., 2000]. Licklider saw the need to move ARPA's contracts from the private sector to universities, convincing his successors at ARPA, including MIT researcher Lawrence Roberts, of the importance of his networking concept.

On the other hand Leonard Kleinrock, who had published the first paper on packet switching theory by July 1961 [Kleinrock, 1961], elucidated Roberts about the feasibility of communications using packets rather than circuits. To explore this possibility, in 1965 Roberts and Thomas Merrill connected a TX-2 computer in Massachusetts to a Q-32 in California with a low-speed dial-up telephone line, confirming Kleinrock's conviction of the need for packet switching.

When Roberts joined ARPA in 1966 he devised his plan for the "ARPANET", a packet switching network. At the conference where he published his work in 1967, Roberts met Donald Davies and Roger Scantlebury from the UK, who had been working on packet switching networks independently since 1964, and who told him also about Paul Baran from RAND[1], working on it as well since 1962. The word *packet* was adopted from Davies' and Scantlebury's work, while the line speed to be used in the ARPANET was agreed to 50 kbps.

In August 1968 ARPA had refined the overall structure and specification for the ARPANET, and in December 1968 Frank Heart from BBN (Bolt Beranek and Newman) was awarded the development of one of its key components: the Interface Message Processors (IMPs). Due to Kleinrock's early developments in packet switching, the first IMP was to be installed at UCLA at the Network Measurement Center, where Kleinrock was working. The second node was provided by SRI, which supported the Network Information Center providing network functions such as the mapping of host names to addresses. USCB and the University of Utah were added afterwards, incorporating application visualization projects such as the display of mathematical functions and 3-D representations over the network. These four host computers comprised the initial ARPANET.

During the following years computers were added to ARPANET at an increasing rate, while work concentrated on completing the host-to-host protocol, which was finished by the Network Working Group led by S. Crocker in December 1970, and named Network Control Protocol (NCP). In late 1971, Roberts asked Robert Kahn from BBN to organize a public demonstration of ARPANET, which would be performed at the International Conference of Computer Communications in 1972. In this same year, electronic mail was introduced: Ray Tomlinson from BBN wrote the first basic e-mail software to send and receive messages, which was to be expanded some months later by Roberts to include selective read, file, forward and reply to messages.

2.1.2 The TCP/IP protocol

Kahn joined ARPA in 1972, which by that time had changed its name to DARPA (Defense Advanced Research Projects Agency). Kahn started working on a packet radio program whose purpose was to develop a protocol for packet radio systems that, built on NCP, would avoid having to deal with

[1] http://www.rand.org/

the multitude of operating systems. But Kahn quickly found out that the NCP protocol was unable to provide the necessary reliability and addressing mechanisms for a packet radio system. In fact, NCP did not even provide end-to-end error control because it was assumed that ARPANET was the only network in existence and that it would be so reliable that no error control would be necessary on the hosts. Packet radio, however, required a reliable end-to-end system protocol that could maintain effective communication while withstanding jamming, interference or intermittent blackout.

Faced with the problem of having a multitude of operating systems, Kahn introduced the idea of an open-architecture network, within which individual networks could be separately designed, offering distinct network interfaces. A common communications protocol would then fulfill the necessary reliability requirements. This work, originally part of the radio packet program, became a separate program on its own, denominated *Internetting* [Gromov, 2000]. Kahn realized, however, that he would need to know the implementation details of each operating system in order to embed new protocols in an efficient way.

In the spring of 1973 Vint Cerf, who had been involved in the original design of the NCP protocol, joined Kahn to work on the new protocol. The association of Kahn's approach with Cerf's experience was soon to give results: the first written version of the new protocol was distributed at a conference at Sussex University in September 1973, and shortly after published under the name of "TCP" protocol [Cerf and Kahn, 1974].

The TCP protocol was intended to support a range of transport services, from the totally reliable sequenced delivery of data (virtual circuit model) to the direct use of the underlying network which might incur in packet loss, corruption, or reordering (datagram model). The first implementations of TCP only allowed the virtual circuit model to take place, which worked fine for remote login and file transfer applications. But further developments during the 1970s, particularly the development of packet voice, made clear that some applications should deal with packet loss directly without relying on the underlying protocol to correct those losses.

For this reason, the TCP protocol was reorganized into two separate protocols: the simple IP protocol providing addressing and forwarding of packets, and the TCP protocol concerning service features such as flow control and recovery of lost packets. This resembles the TCP/IP protocol used today. The User Datagram Protocol (UDP) was added to provide direct access to IP for applications that did not use TCP. The ARPANET host protocol would be changed from NCP to TCP/IP on January 1, 1983 [Leiner et al., 2000] and its relation to military purposes discontinued from that point onward, as MILNET was created as a separate branch and ARPANET became exclusively devoted to research.

2.1.3 Towards a widespread infrastructure

As soon as other research communities realized the usefulness of computer networking, several networks besides ARPANET begun to spring up. The US Department of Energy (DoE) established the MFENet network between researchers in Magnetic Fusion Energy, and the HEPNet for DoE's High Energy Physicists. NASA followed with the SPAN network, while CSNET, a network for the academic and industrial Computer Science community was funded by NSF (National Science Foundation). In general, these early networks were built for special research fields and were closed to other communities. Soon, however, it was realized the opportunity to serve a broader education community, regardless of particular field.

An agreement between ARPANET and CSNET that would allow CSNET to share the ARPANET infrastructure, was the first of a set of initiatives that aimed at converging disparate networks. Another agreement between NSF and DARPA would ensure interoperability between ARPANET and

the NSF-funded networks. Additionally, NSF encouraged its regional, initially academic networks to seek commercial, non-academic users, expanding its facilities to serve them, exploiting the resulting economies of scale in order to decrease subscription costs for all. The NSF-supported networks have grown to form NSFNET, following the intention of establishing a countrywide research network.

Then NSFNET and the British JANET network have announced their plans to serve an entire higher education community, establishing the bridge to Europe. Such was the growth and size of NSFNET, that ARPANET itself would be decommissioned on its 20th anniversary in 1989. Several US Federal agencies also played an important role in shaping a global network. First they started sharing the costs of common infrastructures, such as circuits across the ocean. To coordinate these efforts, the Federal Network Council (FNC) was created. Cooperating with other international organizations such as RARE in Europe, FNC has brought network access to a broader community worldwide.

From an application point of view, one of the major drivers behind the ARPANET was resource sharing. Making a resource available remotely on a single computer was far more economical than duplicating it across several computers. Remote login and file transfer were therefore important applications, but it was electronic mail that had the most significant impact by providing a new model of how people could communicate with each other. E-mail enhanced collaboration, first in building the network, and later between the community using it. Other applications - much of which were precursors of today's technologies such as FTP and Internet Telephony - were not enjoying at that time so much popularity as e-mail did. Nevertheless, they all proved that the network was not designed for a single application alone, but as a general infrastructure on which new applications could be conceived.

2.1.4 The Internet

The early implementations of the original TCP protocol were done for large systems, and when desktop computers first appeared, TCP seemed to be too complex to run on a personal computer. David Clark from the MIT then developed a simple implementation of TCP for the Xerox Alto and the IBM PC personal computers, proving that both workstations as well as large systems could be part of the same network. On a parallel course, Bob Metcalfe at Xerox PARC had already developed Ethernet technology back in 1973 and the 1980s have witnessed the widespread development of Local Area Networks (LANs). These two factors have contributed to a major shift, from a few networks with a small number of hosts to many networks, exploding the scale of computer networking and of its management issues.

To manage the network easier, hosts were assigned names so that it was not necessary to remember their numeric addresses. While initially it was feasible to maintain a single table of all the host names and addresses, the establishment of independently managed networks such as LANs demanded a better solution, which would be the Domain Name System (DNS), a hierarchical and scalable mechanism for resolving host names into their addresses. The software installed at each host also posed some challenges, as well as the propagation of software changes.

Supported by DARPA, UC Berkeley devised the necessary modifications to incorporate TCP/IP into the Unix operating system. In particular, the incorporation of TCP/IP into the Unix BSD system was a major step to disseminate the protocols through the research community, as many began to use Unix BSD as their daily computing environment, e-mail as the communication channel, and FTP for file transfer. During the 1980s, the wide acceptance of TCP/IP has led software vendors to incorporate TCP/IP into their products. Later, the Federal Networking Council unanimously passed a resolution defining the term *Internet* [Leiner et al., 2000]:

"The Federal Networking Council (FNC) agrees that the following language reflects our definition of the term 'Internet'. 'Internet' refers to the global information system that:

(i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons;

(ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and

(iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein."

2.1.5 The World Wide Web

In October 1990, Tim Berners-Lee from the Centre Européenne pour la Recherche Nucléaire (CERN) started working on a hypertext program called "WorldWideWeb". This program allowed the user to add documents and establish links between documents, for example linking text documents or linking an image file to a text document [Berners-Lee, 1996].

With a network interconnecting thousands of devices, desktop computers, and data stores, CERN had the need to search, retrieve and correlate information coming from a wide range of sources and formats [CERN, 1997]. To deal with such an information management challenge, Berners-Lee proposed in 1989 a global hypertext space in which any network-accessible information could be referred to by a single, unique document identifier. The goal was to design an information system that would be able to record random associations - links - between any arbitrary information objects. The power of a link was that it could point to any document or resource of any kind in the universe of information, based simply on a URI (Universal Resource Identifier). Later the term URL (Uniform Resource Locator) was coined, as a URI was used to denote the address rather than the name of each information object.

By the end of 1990, Berners-Lee had developed "WorldWideWeb", a point and click hypertext editor running on a NeXT computer. The initial documents were written using this tool, and published on the first Web server: info.cern.ch. However, as NeXT machines were not so common, Nicola Pellow developed a portable document browser allowing information to be retrieved on any platform [Berners-Lee, 1996]. These developments were released to the general community in the summer of 1991, while the specifications of the HTTP (HyperText Transfer Protocol), HTML (HyperText Markup Language) and URI were published on the server to promote wide discussion and adoption.

The dream behind the Web was of a common information space in which people and machines could communicate by sharing information. Another goal that was never explored to a far extent was that the interaction of persons in a machine-readable information space could produce an accurate account of people work patterns; then machine analysis could provide valuable insight into how persons fit in individually and how they could better work together.

For the three years following the release of the NeXT-based "WorldWideWeb", R. Cailliau helped Berners-Lee in promoting and persuading others to adopt the Web. Several other browsers, such as "Erwise", "Midas", "Viola-WWW", and "Cello" were introduced; the most popular one would be "Mosaic" released in 1993 by Marc Andreesen [Berners-Lee, 1996], that would develop into Netscape. Between 1991 and 1994 the load on info.cern.ch rose steadily by a factor of 10 every year, and by January 1993 the Web comprised already 50 HTTP servers [W3C, 2000a].

In 1994, Berners-Lee founded the World Wide Web Consortium, an open forum where companies and organizations interested in the future of the Web could discuss and agree on new common computer protocols.

2.1.6 The WWW explosion

During the second part of the 1990s the World Wide Web (WWW) enjoyed a tremendous growth as the number of servers increased exponentially. More and more information was available on the Web, attracting more users and institutions, producing in turn an increasing number of Web sites, servers, and host machines connected to the Internet. This loop phenomenon - information availability leading to more information being published - is undoubtedly related with the sense of community that was born around the Web.

In the beginning the network was used by a research community promoting the open publication of ideas and results; that tradition has lived on since then, bringing individuals, schools, governments, companies, and innumerous other organizations and institutions to the Web. Soon, companies started realizing the potential of gathering customers on the Web and started implementing the first forms of e-business: in 1994 it became possible to order a pizza online from Pizza Hut [Zakon, 2001]. But it was only after the release of the Secure Socket Layer (SSL) protocol in the end of 1994 [Treuhaft, 1996] that it became appropriate to charge customers by credit card, allowing general sales to be attained over the Web. This trend would be called e-commerce.

Companies also found 'out that more than buying and selling on the Web, they could implement other business services over the Internet, such as customer service and trading with business partners. This led to the distinction of business-to-consumer (B2C) and business-to-business (B2B) e-commerce. The deployment of business services on the Web has also worked as a forward loop in Web growth, as customers were driven to the Web, increasing the community of users, and the interest from other companies in leveraging such a potential trading channel.

2.2 Market trends and approaches

Soon after the first e-business approaches were put into practice, companies started realizing the potential benefits of reaching such an increasingly growing Web community. But that was not all: companies also realized that they could establish new kinds of trading relationships, what would be called electronic commerce. Relationships with online customers would be regarded as B2C e-commerce, while B2B refers to relationships with trading partners. B2C e-commerce was expected to open new market channels, thereby creating new sales opportunities. B2B e-commerce would allow companies to automate exchanges with business partners, hence accelerating their business processes; it would also allow companies to quickly gather information from different suppliers, thus creating new business opportunities.

B2B e-commerce can have, therefore, significant implications in the way companies conduct their businesses. This explains why there was an order of magnitude distinction between the amount of profit predicted for both types of e-commerce [Timmers, 1999]. It is true that e-commerce has brought a business revolution, but that revolution is not taking place as fast as investors were expecting. After all, such a change in the way companies conduct business could not take place overnight. On the other hand, e-business is not over either, as some proclaim. Business change is undergoing and e-commerce, fueling that change, is finding its own place and opportunities.

2.2.1 The start-ups versus the large companies

The brand-new and far-reaching possibilities of the e-business revolution have created many opportunities from which several innovative business ideas have emerged. The first initiatives had to do with bringing the retail business onto the Web, which would be called *e-tailing*. Amazon.com is an instantly recognized name and an example of such an initiative, but also a name that has lived and survived all the turmoil in recent years. The typical start-up would be a small company, sometimes even arising from a family business, that once placed on the Web would see orders increase dramatically. At least, that would be what its founders would hope. Today there are e-tailers for every imaginable product that can be bought by catalog, from books to clothing, toys, vehicles, or even art. In some cases, there are so many buyers to choose from that there are even some Web sites specialized in comparing products and prices from different sellers[2].

Other start-ups were born from scratch simply as the Web counterparts of traditional companies. This idea has not worked well in every case. The well known case of Boo.com is an illustrative example of how it is not possible to run an e-business just by devising a technologically advanced front-end [Barker, 2000]. Kajsa Leander and Ernst Malmsten, the founders of Boo.com, hoped to launch an e-commerce platform selling sportswear simultaneously in 18 countries. This platform would be built on state-of-the-art technology that would allow the customer to view the sportswear on electronic mannequins from different angles. In a time when everything preceded by an *"e-"* was overestimated, the idea caught the attention of investors, who have willingly financed the launch of Boo.com, which would happen in November 1999. Troubles in controlling costs would begin to arise, as the company did not have a financial director, and the staff lacked retail experience. Boo.com closed a few months later.

These issues were not unique to Boo.com: a study of 400 Web-based companies in Europe [Counsell, 2001] reveals that 72% had taken no advice on the viability of their business and 25% did not conduct monthly cash-flow forecasts.

The successful e-tailers most probably will be existing retailers, with a well-planned Internet strategy. As for other e-businesses, the winners will probably be the ones who already work on the respective business. In other words, technology know-how alone has proved to be insufficient to run an e-business. Even when retailing know-how is present, some e-tailers forget that behind an e-commerce front-end the back-office logistics must be properly planned. In some cases the incoming flux of orders through the Web may be simply overwhelming as illustrated, for example, by what happened in Christmas 1999 in the US [Hollinger, 2000].

By that time, more than 50% of households in the US had personal computers, most of them connected to the Internet, 17% being regular on-line shoppers. Yet, e-tailers were unable to cope with the unusually big amount of orders that customers have placed on their Web sites. As a result, a significant fraction of Christmas presents was delayed. Taking into account that it is in this peak sales season that many people would dare to try on-line shopping for the first time, e-commerce has left a bad feeling of not being able to deliver according to customers requirements. The problem obviously had nothing to do with e-commerce, but with the fact that some e-tailers have failed to appreciate the business they had to build besides setting up and advertising a Web site.

The practice of selling on-line products of services that can be obtained through traditional channels has also led to competition based exclusively on availability and price, because selling cheaper would be the only way to capture customers that could buy the same product on a local shop. In a certain way, this has prevented other companies from coming to the Web, not willing to compete on

[2] e.g. http://www.botspot.com/

such a basis or fearing loss of margins. The truth is that companies do not have to be confined to such market practices, and soon well known brand names have come to prove that a certain quality of product or service demands a certain price, on the Web or otherwise.

That was what happened when larger companies started to embrace the Web. While several new, start-up companies have been born, it was the large companies that made the Internet a broader revolution. Traditional companies, as opposed to start-ups, held industry depth, strong brand identities, and customer trust. They also had better access to investment capital and expertise in managing established businesses, and they were eager to improve business practices and explore potential benefits. Eventually, large companies were soon complementing their business activities with some e-business initiatives. If not for other reason, then for the pressure of seeing other competitors quickly establishing an on-line presence.

For example, for McDonald's UK it was not appropriate to deliver hamburgers to Web users, but McDonald's devised a strategy focused on reinforcing the recreational element of their restaurants [Counsell, 2000]. The Web site invited children to play, join in quizzes or send pictures. An example from the automotive sector, DaimlerChrysler, developed a Web presence focused on showcasing the entire range of its vehicles, and providing after-sales services. DaimlerChrysler aimed at improving their market share on after-sales businesses, which for car companies is typically only 25%. Another kind of service on the Web is provided by some airline companies, e.g. Lufthansa, that besides selling tickets also informs the customer about the arrival status of their flights.

In a short time, almost every big brand was on the Web showcasing, promoting, or selling its products or services. "Almost" because it became so common to have a presence on the Web that some brands, such as luxury items or jewelry, have found detrimental to expose their brand name to such a wide Web audience. After their B2C front-ends were in place, large companies turned to improving their relationship with trading partners by undertaking the first B2B initiatives. DaimlerChrysler started Covisint.com, an exchange portal for automotive parts suppliers, which is being actively used by car manufacturers [Konicki, 2001]. On its turn, Renault invested on setting up an intranet to link its 18 000 employees and to provide them with access to the Internet.

2.2.2 The evolution of B2C e-commerce

B2C e-commerce has grown into a huge range of products and services offered to the consumer. Virtually anything can be bought today through the Web, from expensive items to daily supermarket needs. In fact, the authentication, security and payment warranties are the main issues placing some constraints on business transactions over the Web. These issues, more than imagination, have limited the range of business over the Web. Wherever there is a product or service that can be safely charged to the customer, there is an e-business providing it, as illustrated even in some peculiar forms of e-commerce such as funeral services [Davi, 2001]. The important trends in B2C e-commerce have been therefore the development of payment methods and innovative services to the consumer.

2.2.2.1 e-Payment

The credit card has become the general payment method for on-line purchases and is believed to be used in 98.5% of all e-commerce transactions worldwide [Rodríguez, 2001]. In B2C trading where companies provide products to end customers, this payment method allows the seller to charge an unknown customer before delivering the product. But it also carries some disadvantages for both parties. On one hand, the customer must trust the entity to which the credit card details are being

given. On the other hand, sellers incur in paying a commission split between the bank and the company issuing the credit card. In a traditional shop, commission usually ranges from 1.5 to 3.5% of the purchase value, but for on-line transactions it can reach 4% [Rodríguez, 2001]. This is because the Web is regarded as an immature medium, subject to product returns or late deliveries, which can make purchase cancellation happen more often. The commission also includes a fixed component, rendering small purchase values unprofitable.

These issues have led other payment methods to be taken into consideration. One of such methods is provided by PayPal in the US[3] which specializes in transactions between private individuals, especially useful in on-line auctions. Users can open accounts at PayPal by giving the details of a credit card and bank account. Then PayPal is the only entity to keep the customer sensitive data, taking care of all transactions with other users. Users may also maintain an account balance at PayPal that can be used as a paying resource. PayPal was supposed to work without any commission, subsisting on the interest paid by banks on the accounts opened by users. However, PayPal began to charge a commission for receiving payments.

As an example from Europe, Paysafecard[4] was developed in partnership with IBM Austria [IBM UK, 2001a]. This payment method is one of the simplest ones in its application and provides complete privacy for the customer. Customers can buy a pre-paid card that is available on several shops, and is imprinted with a 16-digit PIN code. The are cards available for several pre-paid amounts, between 20 and 730 Euros approximately. When purchasing over the Web, the purchase value will be simply deducted from the customer's card, by entering the PIN code at the selling Web site. As a further step, IBM Austria is planning to allow customers to recharge their cards over the Web.

Several other payment methods have been devised some of which have already failed due to poor customer adoption [Rodríguez, 2001]. In spite of this, new methods will keep on being developed and most probably there will be an effective alternative to credit cards in the near future.

2.2.2.2 e-Banking and other financial e-services

Of all the services that B2C e-commerce has brought to the customer over the Web, e-banking has been the one that has proved to be more challenging to implement. Nevertheless, there has been at least one bank - SEB from Sweden[5] - that has put the Web at the heart of their strategy [Brown-Humes, 2000]. Aiming at customers who buy shares and mutual funds, SEB has built up an impressive portal for handling personal finances. The underlying assumption is that the average e-customer is more profitable than the traditional one, because they are more likely to trade shares and mutual funds. This assumption has proved to be correct - at least in that economic region.

Other banks - such as the Bank of Scotland[6] - see no need to rush into the Web [Grant, 2000]. The Bank of Scotland is convinced that customers do not perceive the Web as a secure medium and, therefore, do not require a full-fledged online bank. Still, the bank has set up a Web site with limited services, such as opening and viewing accounts and transferring money between accounts of the same bank, as well as applying for mortgages or loans. Another reason behind the bank's slow take-up is the fear of damaging its branch franchising business, which they say is what gives them a broader reach over geographical areas. The Bank of Scotland is most probably just implementing its

[3] http://www.paypal.com/
[4] http://www.paysafecard.com/
[5] http://www.sebank.se/
[6] http://www.bankofscotland.co.uk/

own strategy and considering what is more profitable, because the Web would definitely provide the farthest geographical reach possible.

Other traditional banks, such as Dresdner Bank in Germany[7], BNP Paribas in France[8], or BBVA in Spain[9] have also seen, as SEB did, an opportunity in e-banking. But, as it could be expected from the Web, they are not alone in this business. Internet start-ups have come to take their market share on e-banking as well, the most successful of them being perhaps SMILE from the UK[10]. With a simple and practical Web site, SMILE offers its customers the basic banking services such as managing an account and applying for a mortgage, loan or investment. SMILE has set up a competent customer service and has filled its Web site with precise documentation and explanation of security measures.

Traditional banks have focused on bringing their services onto the Web, and performing them according to their already established name on traditional channels. New e-banking initiatives, on the other hand, leverage on technology know-how and focus on ease of use of a Web site that is supposedly more appealing. This trend can be identified in other e-business areas as well. The market shall dictate which approach will best succeed.

2.2.2.3 The move to the wireless world

The development and success of wireless networks such as GSM (Global System for Mobile Telephony) and the widespread use mobile phones was soon to have an impact on the way people access the Web. Most of that impact, however, is still to come with the implementation of third generation of mobile telephony. The first generation of mobile phones was based on analog technology, with computers allocating bandwidth segments called *cells* - hence the name cellular phones - on the scarce radio spectrum [Cane, 2001]. Most subscribers of this service have moved to the second generation in the 1990s. The second generation retained the same architecture based on cells but now relying on digital technology.

The third generation will dramatically increase the connection data rate from 9.6 Kbps (GSM) up to 2 Mbps with new communication standards such as UMTS (Universal Mobile Telecommunications Service) based on packet-switching instead of circuit-switching. The third generation protocols will allow its subscribers to access new types of multimedia content, and will increase the connecting capabilities of mobile phones to the Internet.

Even within the second generation, initiatives such as the Wireless Application Protocol (WAP) have brought Internet access to handheld devices, making the Web even more accessible. E-businesses have since then supplemented their Web front-ends with WAP accessibility. For example, financial services such as providing stock market data [IBM UK, 2000] or applying for a mortgage [IBM UK, 2001c] were one of the first to provide wireless access. Other applications include e-tailing, such as one electric material supplier allowing its customers to order their products over the mobile phone, possibly directly from the construction site [IBM UK, 2001b]. Wireless access has also proved beneficial for sales force automation [IBM UK, 2001g].

Notwithstanding, WAP acceptance has fallen short of the most optimistic expectations. By mid-2001, only 12 million mobile phone subscribers in Europe were using WAP-enabled devices rather than the expected 250 million [Nairn, 2001]. This may be somewhat related to the low data rate that WAP is able to provide over GSM; in this case, the problem is not WAP itself, but the limitations of the

[7] http://www.dresdnerbank.de/
[8] http://www.bnp.fr/
[9] http://www.bbva.es/
[10] http://www.smile.co.uk/

underlying communication standard. Service providers also undermined the use of WAP by practicing an inappropriate charging method: WAP is charged by connection time, instead of being charged proportionally to the amount of data transferred. As in every typical session on the Web, customers often spend a considerable amount of time collecting and reading information, while expecting the data transfer to be negligible.

WAP is therefore not able to fully comply with customer demands for two reasons: first, the connection is rather slow and second, the service provider charges for the time when the connection is idle. The use of third generation technologies will solve these issues by providing a much higher data rate and based on packet switching, which will relieve service providers from the costs of assigning fixed circuits to customers.

The move towards wireless technology has also expressed itself in other fronts. From Wireless Local Area Networks (the wireless version of common LANs) implementing the IEEE 802.11b protocol to an endless range of devices that will supposedly be able to communicate using Ericsson's Bluetooth protocol[11], there is a trend towards making every device wireless. At the COMDEX 2000 exposition in Las Vegas, as well as at the CeBIT 2001 exposition in Hannover, wireless devices have drawn significant attention from visitors. Mobility and ease of access will therefore shape the customers' use of technology in the forthcoming years.

2.2.3 Characterizing the e-customer

As a business environment, the Web has brought companies in touch with a wide base of potential customers. But it also brought companies closer to their competitors. If it is true that a company can sell a product or render a service to any customer on the Web, it is also true that the customer may choose to buy it from another company, which could be just one click away. However, and in practice, B2C e-commerce does not always take place with this simplicity. On one hand, there are companies that only sell their products or perform their services to a restricted set of customers, for example inside a single country or region. On the other hand, customers often buy from a company that, even with a global presence on the Web, is in some sense closer to them, for example because of shipping expenses or because they speak the customer's mother tongue.

Even taking into account business restrictions or consumer behavior, the Web is a significantly different medium from the traditional marketing channels, and companies have to strive to obtain the customer's preference. With competitors just one click away, a brand or Web site is increasingly vulnerable. Within the blink of an eye, it could be a competitor gathering the customers' attention by means of the most attractive or feature-rich Web site. In an environment that is constantly being taken over by new and improved products or services, business can become quite uncertain.

To survive in this business jungle, there are thus not many choices than to aim at customer loyalty through inspiring a high level of trust and ensuring customer satisfaction accordingly. The customer is the one to decide what, how, when and how much he or she is buying. In some cases, the customer even specifies the desired product by customizing over several options provided by the seller, as for example in the Dell Computer store[12]. The customers will also expect to be able to know the status of their orders, and to receive exactly what they ordered for the agreed price without unexpected delays. E-customers are led into suspicion and distrust easier than their traditional counterparts, as they place their orders without any personal contact. The e-customer is therefore highly demanding, and unforgiving.

[11] http://www.bluetooth.com/
[12] http://www.dell.com/

E-customers are also able to influence other potential customers in the Web community, and they can gather support for their complaints. In effect, there are even some "opinion portals" where customers can express their complains on-line [Adolph, 2000], since it is typical for an e-customer to search for the opinion of others before deciding for a product or service. In general, and compared to their traditional counterparts, e-customers can leverage their community to better serve their purposes. Curiously, but perhaps not very surprising, all these issues of trust, demand and ability to leverage from e-communities play a role on B2B e-commerce as well.

2.2.4 The evolution of B2B e-commerce

There are several reasons for considering B2C and B2B e-commerce separately. In fact, they are distinct forms of e-business. B2C e-commerce has to do with the deployment of Web resources that enable business transactions with the end consumer. It focuses on catalog presentation, order processing, and e-payment systems. On the other hand, B2B e-commerce encompasses any initiative that allows businesses to trade over the Internet or over a similar technological infrastructure. It is dominated by high value transactions, trust relationships, complex business practices, data communications, security, and a multitude of interaction models. This means that B2B e-commerce brings, besides the B2C requirements, additional challenges of its own.

B2B e-commerce also exhibits a different growth pattern when compared to B2C. A significant fraction of consumer buying is seasonal. Christmas, the beginning of the school year, or an important sports game, are examples of events that drive an unusual amount of consumers to the Web. These events represent sudden jumps in B2C e-commerce activity. But B2B e-commerce, on its turn, grows more steadily as companies gradually realize the opportunity of using the Internet for their benefit. Each time a company decides to use the Internet for trading with business partners, B2B e-commerce goes one step further. This explains perhaps why B2B was predicted to become an order of magnitude bigger than B2C e-commerce.

2.2.4.1 Electronic data interchange

The underlying rationale for B2B e-commerce seems to precede even the Internet. In fact, some authors [Unitt and Jones, 1999] claim that it is not actually a recent paradigm, but something that companies have already been doing since the times of Electronic Data Interchange (EDI). If EDI can be defined as the exchange of structured business data in an agreed standard format between computer systems of trading partners, then it surely falls within the scope of B2B e-commerce.

EDI was introduced in order to provide faster trading cycles, reduced costs, and error reduction. The most widely accepted EDI standards were to be ANSI X.12 in the US, and UN/EDIFACT in Europe. These standards define the message format that EDI systems must follow in order to be able to operate together. Most EDI users must connect to each other over a dedicated, proprietary Value Added Network (VAN), which acts as an intermediary between trading partners, providing message storing, audit trails, and solving mismatches regarding communication protocols.

But VANs require a dedicated and proprietary infrastructure that is difficult and costly to implement. Eventually, and despite the fact that these standards have been promoted by governments and industries, EDI is still difficult to adopt, and it is estimated that it has attracted only around 100 000 commercial companies worldwide [Unitt and Jones, 1999]. It is also believed that some companies have adopted EDI by being coerced or pressured from larger, more powerful business partners who had already adopted it [Ratnasingam, 2001].

2.2.4.2 Internet technologies

The Internet has changed this scenario. Instead of having to set up proprietary and dedicated networks, companies can now use a common and widely accessible infrastructure. And by making exchanges easier and cheaper, the Internet allows more companies to endorse EDI. But with the Internet, new technologies have also come into play, namely the Extensible Markup Language (XML) [W3C, 1997].

XML has been used as the foundation to develop other specifications that rival EDI. The Financial Information Exchange Protocol[13], RosettaNet[14] or BizTalk[15] are only a few examples of XML-based frameworks that aim at specifying messaging protocols between business partners. Typically, these frameworks assume a particular message sequence and define in detail the format of each message in that sequence[16].

But these frameworks have not been focusing exclusively on developing alternatives to EDI. There are also initiatives that try to leverage the some or all of the effort that has been put on EDI in the last decades. For example, the Open Buying on the Internet framework[17] is not XML-based but EDI-based. Additionally, there are standardizing efforts being taken, such as ebXML[18], XEDI[19] and XML/EDI[20] that are trying to map EDI standards to XML, thus paving the way for the use of EDI over the Internet. As a consequence, EDI lives on and XML is not necessarily a competitor technology.

Another key feature of the Internet is that it is a ubiquitous and widely accessible infrastructure while the traditional VAN was confined purely to its members. This means that by using the Internet, companies are given an expanded universe of potential business partners. That is, the Internet is not just a global infrastructure, it can be a global marketplace where business partners meet and perform trade. This idea led to the concept of *e-marketplaces*.

2.2.4.3 E-marketplaces

An e-marketplace is a Web portal where buyers and suppliers come together to explore new business opportunities. Most e-marketplaces focus on the trade of parts, materials or goods either within a single industry sector (vertical e-marketplace) or across several industry sectors (horizontal e-marketplace). There are three main types of e-marketplaces [Norris and West, 2001]: the *seller-driven market* is an e-marketplace promoted by a consortium of suppliers who place offers within the same industry or service sector, the *buyer-driven market* is maintained by a group of buyers who aggregate purchase needs so as to achieve advantageous conditions when buying from suppliers, and the *open market* is an e-marketplace owned by an independent third-party.

The goal of an e-marketplace is to attract the biggest possible number of buyers and suppliers, which will become members of that e-marketplace. Buyers bring purchase needs, while suppliers bring selling offers. Every e-marketplace thus comprises two main areas, commonly referred to as the *supplier side* and the *buyer side*. The supplier side is the entry point for suppliers, where they can advertise their selling offers; in the buyer side, buyers advertise purchase needs. Usually, e-

[13] http://www.fixprotocol.org/
[14] http://www.rosettanet.org/
[15] http://www.biztalk.org/
[16] B2B frameworks are discussed in detail in chapter 5.
[17] http://www.openbuy.org/
[18] http://www.ebxml.org/
[19] http://www.xedi.org/
[20] http://www.xmledi-group.org/

marketplaces provides additional services such as auction (where a supplier sells to the buyer who submits the highest bid for a given selling offer), and reverse auction (where a buyer buys from the supplier who submits the lowest bid for a given purchase need). The e-marketplace is the central hub providing all these services.

Whenever a buyer finds an interesting selling offer, or whenever a supplier is interested in fulfilling to a given purchase need, the entity that runs the e-marketplace will work as a broker, mediating the contact between buyer and supplier. Buyer and supplier will typically interact solely through the e-marketplace, and never directly. The broker will usually charge a monthly fee or a commission on the transaction value. On one hand, the presence of the e-marketplace as an intermediary ensures that the supplier will be paid for the delivered goods, even if the supplier does not know if the buyer is trustworthy or not. On the other hand, since buyers and suppliers cannot interact directly, they cannot negotiate or develop long-term relationships.

Another pitfall of e-marketplaces is that by doing auctions and focusing on the buying and selling of catalogued products, they expect purchasing decisions to be based mainly on price. This does not provide compelling motivation for other companies to join the e-marketplace, as they may fear price erosion [Veeramani and Talbert, 2001]. In conclusion, B2B frameworks and e-marketplaces will have to mature in order to deliver all the advantages of B2B trading over the Internet. Meanwhile, the e-business revolution goes beyond B2C and B2B to include other manifestations as well.

2.2.5 Other forms of e-business

2.2.5.1 Public services and B2G

By the time B2C e-commerce was already active on a large scale and virtually every business sector had some form of online presence, the Web became appealing for public services as well. As the Web was increasingly becoming a wide and low cost, accessible infrastructure for the whole community in general, it was becoming an undeniable part of everyday life.

Public services then realized the opportunity to drive thousands of citizens away from their offices by providing at least some part of their services through the Web. The technological feasibility of this approach had already been proved by B2C e-commerce front-ends that were capable of handling thousands of customer interactions per day. In the case of public services, with well-defined and highly bureaucratic processes, becoming online would relieve employees from a lot of paperwork. For the ordinary citizen, having public services accessible online could avoid spending hours on queues.

An example is the *Caisse Allocation Familiale* (CAF) in France[21], a family allowance office that handles child benefits, childcare allowances and even student accommodation grants, that are regularly claimed by 10 million French families. Despite having several points of contact with the general public through call centers and branches throughout the country, CAF launched a Web site allowing citizens to perform some of the operations online, including assessing the impact on their income of applying for different types of benefits. Such operations would previously required a personal visit to a branch office, where staff would access that information from a central database. By making the database accessible through the Web, the CAF's staff (which amounts to 20 000 employees) is relieved from these routine tasks while the citizen is spared a trip to the branch office.

[21] http://www.caf.fr/

Besides public services, different possibilities for exerting civic duties have emerged too. For example, recent elections in Belgium have used two different methods of recording votes: while the majority used the traditional voting papers, 44% of the electorate voted using a PC [IBM UK, 2001e].

The Web has brought new opportunities for governmental practices as well and, in particular, is allowing governments to negotiate and place online their purchasing needs. A pioneering experience was perhaps that of September 2000, in which the US federal government ran an online reverse auction for a huge number of IT products using an e-marketplace[22], which was estimated to have led to significant savings.

Since then, online marketplaces have been used for other government purposes too, including military. As an example, again from France, the *Direction Generale de l'Armement* (DGA) which is responsible for supplying armed forces with equipment, was reported as having plans to launch an armaments portal [Lancesseur, 2000]. This portal would become known as Ixarm[23], providing reverse auctions and support for EDI catalogue and government supply transactions. Collectively, initiatives such as public e-services and online marketplaces supporting government needs have been categorized as another variant of e-commerce called B2G (business-to-government).

2.2.5.2 B2E, e-learning and e-recruiting

In parallel with these developments, and since software products were increasingly becoming Web-enabled, companies started using the Internet and Web technologies for their internal benefits, with special emphasis on the management of human resources. Companies have realized that what the Internet does for individuals and organizations - expanding the kind and amount of information available - it could as well do for employees.

Networks under the form of *intranets* and employee services applications can improve communication and collaboration, and accelerate the dissemination of information throughout the enterprise. Employees and managers become able to access and update personal data, or paycheck processing, or procedures. Gradually, and by using those information services, employees will become managers of their own data. So besides providing information at lower cost, the implementation of systems using Internet and Web technologies can reduce traffic in administrative departments and significantly improve the performance of information-related processes.

This trend has been generally classified as B2E (business-to-employee), although it is arguable whether this is a form of e-commerce or not. Other Internet-related technologies, not just intranets, can also play a role in B2E. For example, IBM EMEA (Europe, Middle East and Africa) was reported to be implementing a solution providing employees with WAP-access to e-mail, calendar, and internal directories while they are out of the office [IBM UK, 2001f]. The fact that information plays such a central role is a consequence from the fact that today's "new economy" is based on knowledge. The upcoming perspective is that the value of an organization lies in its intellectual capital, not on financial variables.

However, even knowledge is exhibiting a faster life cycle, requiring staff to be regularly trained on new skills. Because staff has to excel in their jobs for a company to gain or sustain competitive advantage, it is necessary to improve or augment the competencies of employees. Companies are finding that the Web may help in corporate training with some advantages when compared to traditional methods [Kühn, 2001]. Employees can receive training right on their workplace without being

[22] http://www.buyers.gov/
[23] http://www.ixarm.com/

absent. The company also avoids the costs associated with trips that employees would take to receive training.

This trend is usually called *e-learning* and schools and universities are also taking advantage of it. Some academic institutions have started to use the Internet to enhance communication between the academic community while developing the students' skills to work with it. Others have developed internal information infrastructures, such as promoting the use of laptops in conjunction with wireless networks to provide new, time- and place-independent ways of learning [IBM UK, 2001d]. Still others actually deliver educational content over the Internet [Niedercorn, 2000].

Finally, a human resources related area that deserves attention is *e-recruiting*, as several job- and career-related portals are available[24]. Usually companies or their headhunters can advertise a place or vacancy by paying a certain fee, while candidates can freely answer or submit their curriculum.

2.3 Trends in Supply Chain Management

The Internet, the Web, and e-business in general have come as unavoidable catalysts to business practices. From B2C to B2B, e-commerce has demonstrated its potential to support and enhance existing trading activities or even foster new types of relations with both customers and suppliers. For the supply chain, some of these possibilities represent alternative and advantageous ways of implementing existing approaches. But they also encourage innovative ways of conducting business that were previously unavailable.

Having a globally connected and accessible communication platform has brought challenges on three levels[25]:

- first, and from the example of many companies that have endorsed a B2C e-commerce strategy, there is a new market channel to be exploited,
- second, the Internet and the Web provide a channel for communicating also with suppliers allowing, in general, to improve collaboration between business partners,
- and third, because all business partners can share the same infrastructure, it becomes possible to globally manage the flow of information and coordinate actions across the supply chain.

Opening and exploiting a new market channel can be considered as falling within the scope of B2C e-commerce. Interaction and collaboration between business partners could be placed within the scope of B2B e-commerce. But supply chain management brings the additional challenge of coordinating a wide set of business partners, bringing together and correlating information available from each one of them in order to devise the proper flow of products that will ensure the best supply chain performance.

2.3.1 Increased intimacy with business partners

For years, consumers and suppliers in the supply chain have been devising ways to coordinate their actions in order to lower costs while satisfying increasingly demanding customers. One of the first approaches towards that aim, perhaps the most illustrative one, is Just-in-Time (JIT) manufacturing

[24] e.g. http://www.stepstone.com/

[25] adapted from [Lescher, 2000]

[Damodar and Carol, 1991]. In this approach, inventories would be kept to a minimum while raw materials or components were expected to arrive on the right moment, when production would need them. This required manufacturers to inform their suppliers of production plans, not just about material needs as happened before.

The introduction of the Internet and its associated technologies has fostered the development of other approaches towards improving supply chain performance, which have resulted in the development of collaborative relationships either to the supplier side or to the buyer side, or both [Anthony, 2000]. Initiatives such as these are resulting in an increased partner intimacy, as suppliers and customers share more of their internal information for the benefit of the whole supply chain.

2.3.1.1 VMI

In some situations which require maintaining a certain level of stock (e.g. retailing), techniques have been developed such as Consigned Inventory or Vendor Managed Inventory (VMI) [Hall, 1998]. In this case, suppliers are responsible for maintaining the stock of their products at the retailer's site. The retailer pays for the goods only after having sold them, and provides the supplier with sales predictions. This information, together with the desired stock level, enables the supplier to manage the retailer's stock. Since the retailer only pays when the goods are sold, the costs associated with surplus or overstock are passed on to the supplier. However, since the retailer provides the supplier only with sales predictions, inaccurate market forecasts can undermine the correct operation of VMI. This is unavoidable to a certain degree, as market behavior is often unpredictable.

2.3.1.2 CPFR

Customer and supplier were therefore led to devise ways to handle forecast exceptions, such as what happens in Collaborative Planning, Forecasting and Replenishment (CPFR) [Anderson, 2000]. In this case, retailer and supplier first agree on a business plan, deciding the products to be jointly managed and establishing performance indicators. Then they develop a single forecast of end-consumer demand based on analysis of previous sales and the promotion calendars for the new period. This forecast dictates the first amount of orders to be placed to the supplier. During the sales period, actual sales are compared to the initial forecast and any exceptions are handled according to the initial agreement. The forecast is then updated regularly before new orders are generated. From these, the short-term orders are immediately placed, while the long-term ones are used for planning.

2.3.1.3 Scan-based trading

In this approach, supplier and retailer use a shared database containing agreed-upon prices for the supplied products [Caldwell, 1999]. The usage of this common database prevents supplier and retailer from maintaining separate price lists. This database may be hosted by one of them or by a third party. In a similar way to VMI, the supplier owns and manages the retailer's inventories for the supplied products. The main feature is that those products are paid when they are being sold to the end consumer, precisely when the product is being scanned at the point-of-sale (POS), hence the name *scan-based trading*. The daily POS scanner data is used to look up prices on the shared database and paying the products to the supplier.

2.3.1.4 Procure-to-pay transactions

Procure-to-pay concerns a set of traditional procurement transactions from placing a purchase order to paying the delivered product. This is referred to the *ordering* phase. Additionally, procure-to-pay comprises two more phases: *sourcing* and *analysis* [Punzak, 2001]. Sourcing takes place before the ordering phase and deals with the development of requests for quotation (RFQs), contract negotiation and award. This will result in a chosen supplier for the ordering phase. Analysis takes place after the ordering phase is complete, and its purpose is to evaluate the performance of the supplier and attain market assessments. If deemed appropriate, a sourcing phase will take place before a new ordering phase. The procure-to-pay business cycle - sourcing, ordering, analysis - used to be scattered over a set of internal business activities. Gradually, it is being undertaken as an end-to-end business process with the collaboration of external partners during each phase.

2.3.1.5 Catalog management

The purpose of this initiative is making buyers and suppliers share information about the products they purchase or sell, keeping the information synchronized across several suppliers and customers [MRO, 2002]. Additionally, it aims at providing the information for customers to place orders against new products with minimal data entry. Catalog management is a way to enhance awareness of partners' products and improve the ordering processes.

2.3.1.6 Outsourcing

Delegating a set of business processes to a third-party results in establishing a collaborative relationship where an external partner handles a supplier's products or services. A typical example is outsourcing the fulfillment of goods to a third-party logistics provider. This provider handles the entire finished goods inventory and, when the items are purchased by a retailer, it attaches the label, packs the products, ships and delivers them. The supplier provides sales order information and receives shipment details from the third-party logistics.

Outsourcing can be applied to other business processes as well. Such is the case of order-to-cash functions [Allen, 2002], which comprise key activities in the financial supply chain such as credit analysis and approval, invoice and billing, funds application, dispute resolution, financial analysis and reporting. These are complex and labor intensive processes, and generally not belonging to a company's core competencies.

2.3.2 Automating and exposing business processes

Most of today's interactions between enterprises are attained manually. From a simple request for proposal to acknowledging the delivery of goods, many business activities cannot be carried out without the phone, fax or, more recently, the e-mail. This practice is inefficient because manual operations are costly and time-consuming, but not only. Manual operations also make use of a multitude of resources and communication mechanisms that are far from providing a uniform and convenient infrastructure for business exchanges. An example is that the phone is enough to inquire about the status of a particular order, but the fax is needed to place an order document, and often the process initiated by a phone call cannot proceed without a sending a fax as well.

The Internet and its associated technologies have the potential to change this practice by providing a ubiquitous and inexpensive networking platform for attaining all kinds of business exchanges. All

the desired information, plus any required documents, can be exchanged through a widely accessible information highway, providing enterprises the ability to interoperate with just about any potential trading partner. Moreover, Internet technologies may connect human resources, but they are most effective in connecting applications and information systems. Information moving inside the enterprise, as well as data from external trading partners, may now cross the enterprise borders through a networking platform that can be easily integrated with back-end information systems. As a result, it becomes possible to synchronize, integrate and automate business exchanges between trading partners.

2.3.2.1 Application-level automation

On the Internet, XML has become one of the preferred ways to exchange information. At the same time inside corporate intranets, where Internet technologies are widely employed, XML became an attractive alternative for representing system-independent data. The Internet protocols also encourage message-based interaction patterns (as opposed to programming interfaces), making XML an excellent technology for formatting, transferring and processing information. It is not surprising, then, that XML has become central to enterprise application integration (EAI), both within and across enterprise boundaries.

There are several advantages that explain the success of XML, namely:

- XML provides flexibility in defining document formats;
- it is easy to create XML documents, whatever their format;
- XML is suitable for existing Internet and Web protocols;
- it is easy to write applications that process XML documents;
- XML is both formal and concise;
- XML can be both machine- and human-readable;
- there is a plethora of tools for manipulating XML;
- XML is vendor-independent.

An additional advantage is that it is easy to transform one XML format into another. In fact, a particular technology called XSL Transformations [W3C, 1999b] has been proposed in order to automatically convert one XML format into another. This feature is extremely important for B2B interactions since it allows business documents to be efficiently converted between trading partners using different formats. This possibility was the ground for new mapping services [Hildreth, 2001] aiming at translating XML documents.

Still, not all business information is available under XML documents so a more general solution is often required for retrieving data from other sources and sending it to other targets. This is precisely what some commercial solutions have attempted, such as the one from CommerceRoute [CommerceRoute, 2000]. This product is able to retrieve content from and send content to remote databases, file sharing protocols, and Internet protocols, providing a graphical editor for defining document transformations.

Internet technologies such as XML and XSLT are excellent for translating business information, besides being suitable for data exchange over Internet protocols. Some situations, however, require more than syntactical agreement. When the meaning of an information field is unclear or ambiguous, a data dictionary is necessary in order to correctly handle that information. This is precisely what is provided in RosettaNet[26]. This B2B framework comprises two dictionaries: a *technical dictionary* concerning product information and a *business dictionary* defining the language of information items such as orders, shipping and payments. The two dictionaries ensure that trading partners (or their applications) clearly understand each other when they refer to specific products or information.

2.3.2.2 Business-level automation

Once applications are able to operate according to specific semantics, trading partners must focus on how to manage exchanges between them. Each partner has its own business processes and their interaction comprises one or more synchronization points between those processes. According to traditional practices there is usually a dominant partner or contractor responsible for managing the interaction process. Other partners are secondary and do not have visibility of the entire process; they react in response to the process being managed by the principal participant. Control, as well as the opportunity to improve the process, lies solely with the main contractor. None of the participants has a global view into how the process is being carried out or into how they can improve their overall performance.

Increased connectivity and easier application integration are promoting decentralized ways to manage these interactions. Partners will first define a shared process that will conduct their exchanges. This represents the exchange they have agreed to undertake. Additionally, this process becomes the ground for evaluating performance and fine-tuning their exchanges. Through a shared process each partner can expose elements of its process domain while limiting visibility into internal processes. This concept has been discussed in e-business literature [Yee, 2000] and it has also been brought up as formal basis for inter-organizational workflow management [Aalst and Weske, 2001].

Some commercial solutions provide functionality to manage B2B exchanges at a business level. Document mapping solutions usually provide some kind of support for defining and managing B2B processes. For example, the product from CommerceRoute [CommerceRoute, 2000] mentioned in the previous section contains a graphical editor for defining processes as well as scheduling them for one-time or regular execution.

Driving B2B exchanges according to process definitions is also the aim of RosettaNet. The framework is based on Partner Interface Processes (PIPs) which specify how two processes running on different enterprises should be standardized and interfaced. A PIP is actually an XML document that defines how a set of B2B exchanges should be conducted. RosettaNet proposes PIPs for a wide range of business activities: product information, order management, inventory management, marketing information, and customer service and support; most of these PIPs were defined with Information Technology and Electronic Components sectors in mind.

2.3.3 The role of enterprise information systems

Automating B2B processes assumes that the integration of internal processes has been tackled already. In fact, on a parallel track to the uprising of e-business, recent years have witnessed the development

[26] RosettaNet is described in more detail in section 5.2.6 on page 200.

of comprehensive information systems for enterprises, that had an enormous impact on the integration of internal business processes.

From the early days of MRP (materials requirements planning) systems aiming at minimizing inventories while maintaining adequate materials for the production process, through MRP II (manufacturing resource planning) that added production planning and control, and DRP (distribution requirements planning) that applies MRP principles to distribution, enterprise information systems have evolved into ERP (enterprise resource planning). ERP systems support most major operational processes such as [Davenport, 2000]:

- financial and accounting processes, including treasury, accounts payable and receivable, investments, and financial reporting;
- manufacturing processes (often with some help from specific shop-floor systems);
- trading processes such as sourcing, procurement, shipping, billing, and payment;
- order fulfillment and customer service processes;
- sales force management;
- human resources management;
- maintenance of plant and equipment;
- construction and project management;
- management processes such as reporting and analysis.

A typical ERP system comprises several modules usually divided according to major processes, and they usually provide a process-oriented structure of work. The flow of information across the enterprise is orchestrated by the system, which has both advantages and disadvantages. On one hand, the ERP vendors claim their systems are based on industry best practices, so that any enterprise will supposedly benefit from them. On the other hand, that leaves almost no room for deviation from procedures that the system enforces. ERP vendors also claim that their systems can be highly customized to better fit enterprise needs, but the fact is that changes from a certain point are not supported by the vendor and will be prone to problems on system upgrade.

Notwithstanding, ERP systems, whenever installed successfully, significantly improve internal business processes. Even some processes that were hidden among series of ad-hoc activities become controllable. In fact, installing the system requires a level of documentation of business practices and selection of configuration options that increase the awareness of the enterprise about its own processes. The intended result is a customized, comprehensive and integrated information system covering most aspects of its business activities.

The usage of ERP systems has thus been most of the time an inward effort towards the improvement and integration of internal business processes. In recent years, however, two major factors have granted enterprise systems an increasingly important role in managing the supply chain and trading relationships. One was the development of best-of-breed solutions specialized in planning, scheduling and optimizing the supply chain. The other was the evolution of ERP systems towards Web-accessibility and growing use of Internet for connecting internal and external resources.

2.3.3.1 All-in-one versus specialized packages

ERPs are comprehensive information systems that can be customized, within certain restrictions, to support a large set of business processes. But in order to provide a generic solution that is suitable for several enterprises, ERPs focus on a set of business processes that are common to most enterprises from a given industry or business sector. As a consequence, ERP systems cannot address specific, one-of-a-kind processes that enterprises often require. This leaves an opportunity for other software vendors (or the same ones!) to come up with specialized, best-of-breed solutions targeted at particular needs. A significant part of these solutions is concerned with supply chain management and constitutes a group of tools generally called Advanced Planning and Scheduling (APS) [Berger, 1999]. Under APS the following business functions are addressed:

- demand forecasting: aims at statistically predicting future sales taking into account historical sales data and marketing data;
- sales planning: aims at planning sales operations based on predicted demand;
- inventory planning: aims at finding the optimal compromise between inventory levels and customer service level;
- production planning: aims at scheduling the production according to production capacity and availability of materials;
- distribution planning: aims at optimizing the use of resources by choosing the appropriate locations, distribution routes and inventory levels, while minimizing transportation costs;
- shipment scheduling: aims at determining the optimal timing and contents of each shipment while complying with delivery dates;
- inter-enterprise collaboration: aims at supporting collaboration with customers and suppliers by enabling exchange of data such as product design or customer demand.

Available APS tools focus on supporting one of these goals while being integrated with the local ERP or legacy system. An illustrative example is the Advanced Planner and Optimizer (APO) module from SAP [Knolmayer et al., 2001]. Directly connected to the SAP R/3 main system, APO is capable of analyzing demand, scheduling production and planning distribution. It also aims at supporting collaborative planning and it provides a *supply chain cockpit*, a graphical tool for visualizing and monitoring supply chain activity.

In order to provide some of these features, APO works in close connection with two additional packages. One is the Business Information Warehouse (BW) [Becker et al., 2001], a data warehousing solution [Chaudhuri and Dayal, 1997] for accessing both internal and external information, with sophisticated tools for analysis and reporting. The other is the Logistics Execution System (LES) [Knolmayer et al., 2001] aiming at inventory management and transportation; this solution is supported by wireless barcode scanners used to track merchandise in picking and put-away operations, updating its location directly on the main system.

Other packages aim at optimization tasks such as finding the optimal production and distribution strategy in order to minimize inventory and transportation costs. For example, given the products of a company, their net profit, and the availability and cost of resources to manufacture those items, the goal is to find the optimal amount of each product to manufacture in order to maximize profit. Another

example, given the number of distribution centers and their exact locations, as well as transportation costs to those locations and from those locations to customers, the goal is to find the optimal level of inventory at each location in order to minimize transportation costs.

These challenges can often be reduced to operations research problems which can be tackled using mixed-integer programming and multi-objective optimization techniques [Shapiro, 2000]. Such is the case of SLIM/2000 [Slim, 2002], an off-the-shelf modeling tool for supply chain planning. This package is able to import and export data from typical spreadsheets and databases, and to create and optimize linear and mixed-integer programming models based on those data. It includes utilities for generating transportation networks and can be integrated with geographical information systems (GIS) to display maps of data inputs and outputs.

2.3.3.2 Enterprise systems meet the Internet

The rise of the Internet has had (and is still having) a strong impact on enterprise information systems. On one hand, ERP systems had to be integrated with Internet technologies which have pervasively influenced work and business practices. On the other hand, additional or improved specialized packages have been developed, which take advantage of increased connectivity and information accessibility, or even attempt new trading patterns over the Internet.

The move which made ERP systems and the Internet collide has probably begun with the simple Web presence most companies have established for advertising their products. The product catalogue was developed and managed internally on their information system, and product information replicated on the Web site. Gradually, the site was redesigned to improve presentation, add up-to-date information, enhance search and linking, and introduce interactive capabilities. Soon, companies would be attempting to provide customer service or handling customer orders through the Web site, and suddenly front- and back-office had to be integrated. For the ones that have succeeded in this task, then e-business capabilities could be extended not only to the customer side but also to the supplier side. Finally, with these relationships in place, the enterprise begins to see new ways of running its own business and starts improving their systems and processes accordingly.

This presumed evolution shows two opposite and simultaneous directions for integrating enterprise information systems [Kalakota and Whinston, 1996]: one is the B2C integration of advertising, sales and support service activities with customers; the other is the B2B integration of procurement, distribution and logistics activities with suppliers. Both scenarios require the integration of Internet front-ends with the enterprise information system. From the two possible types of back-office integration - batch-oriented and real-time [Lamond and Edelheit, 1999] - only the latter suffices supply chain management requirements. This explains why enterprise information systems must be integrated with Internet technologies, either through ERP system improvements or by means of specialized packages that grant those capabilities.

Realizing this requirement, SAP was one of the first ERP vendors to endorse an e-business approach, positioning itself as an e-business solution provider with the creation of mySAP.com. According to its original conception, mySAP.com was divided in two main components: the Workplace and the Marketplace, both Web-based. The Workplace is an internal portal where users are provided with more convenient and easier access to the applications needed for their tasks, using the Web browser alone. The contents and appearance of the Workplace are customized for each user. The other component - the Marketplace - provides access to a set of external e-marketplaces hosted by SAP and focusing on a wide range of business sectors. Through the Marketplace the user may find suppliers, request quotations, place orders, and exchange documents with other partners.

Later, an alliance between SAPMarkets and Commerce One reshaped mySAP.com into two other components: MarketSet and Enterprise Buyer. The first supported collaboration, inventory strategies like VMI and CPFR, and the setup of e-marketplaces. The second was mainly targeted at e-procurement and purchasing activities. Application hosting solutions were also provided. More recently, and under different names, mySAP.com has been extended to include new components such as human resources management, financials and customer relationship management. As mySAP.com continues to grow, including more and more functionalities, it is becoming a Web-based ERP system taking advantage of the new possibilities of e-business by allowing enterprises to collaborate and perform trade over the Internet.

2.4 Reengineering and re-organization

Enterprise resource planning systems focus on improving business processes by providing an enterprise-wide information infrastructure and employing best practices and methods of work that have worked for other companies. These best practices have been obtained by documenting business processes and inserting appropriate changes for improving them. ERP systems then bring the necessary information technology functionalities to support those processes [Scheer and Habermann, 2000]. Enterprise information systems, thus, are enablers of business process improvement.

In general, information technology (IT) enables business processes to be restructured and automated to take advantage of efficiencies in information gathering, storage, processing, retrieval and presentation [O'Neil and Sohal, 1999]. This fact has fostered significant developments in enterprise integration architectures [Bernus et al., 1996] and has given rise to paradigms such as Business Process Reengineering (BPR) [Grover and Malhotra, 1997]. These paradigms aim at reengineering business processes so as to improve efficiency in an increasingly competitive business environment.

In particular, the main challenges at the beginning of the 1990s were [Vernadat, 1996]: meeting customer requirements, reducing the time-to-market of products, and manufacturing products at low cost with increased quality. No longer was it enough for each employee or department to perform their job as well as they could. Each business unit should become aware of the role of other resources along the process, so as to perform their job in a way that decreases overall cost, avoids delays, and maintains product quality. Process reengineering is about becoming aware of current business processes, realizing their inefficiencies and bottlenecks, and redesigning those processes in order to improve end-to-end performance.

But from ordering raw materials to delivering a finished product there is a myriad of business activities taking place such as marketing, manufacturing, logistics and accounting, to name just a few. Some of these may characterize the enterprise and its business, where the enterprise applies its particular know-how (e.g. in manufacturing), while others are auxiliary or secondary tasks that must be performed in all cases and regardless of what the main business is (e.g. accounting). The first are called *core competencies*, while the second can be referred to as non-core competencies.

Core competencies and their corresponding business processes are controlled by internal resources; its improvement and efficiency are subject to existing know-how in the enterprise. Non-core business processes, however, cannot be optimized beyond a certain point because they do not belong to the enterprise's expertise. Reengineering these processes requires the company to rely on external consultancy, or in other words, relying on those for whom those processes are core competencies. Managing those non-core processes becomes costly and cumbersome, and the enterprise may prefer

to outsource those processes instead of performing them in-house, so as to devote its resources to what the enterprise does best.

As a matter of fact, focusing on core competencies leads to a natural increase in outsourced business activities, not only because enterprises are willing to outsource non-core business processes, but also because there is an increasing number of available service providers who specialize themselves as well. An example is that of Fedex which, from delivering merchandise, has turned into a logistics provider managing the whole inventory and shipping of its clients[27].

A paradigmatic case of focusing on core competencies is that of Cisco Systems, which has taken outsourcing to unprecedented proportions [Kraemer and Dedrick, 2002]. Cisco, cited as being the largest networking equipment company in the world, works with a wide range of business partners. These include resellers that sell and support Cisco products, network specialists providing network integration and manufacturers that represent Cisco's production capacity. Cisco itself does not produce or sell its products. In a highly dynamic technological environment, Cisco has placed its focus on a product innovation strategy, directing its own resources to research and development (R&D) and acquisition of other companies developing key technologies. Therefore, Cisco relies on external partners for competencies in operations and customer service, and was able to grow rapidly without heavy investments, particularly in manufacturing capacity.

The success of Cisco's strategy was also supported by extensive use of the Internet and e-commerce [Kraemer and Dedrick, 2002]. After installing an ERP system, Cisco kept on developing information systems to support several business processes. At the same time, it was selling its products online, which would eventually account for most of the sales. Gradually, Cisco was attempting to integrate the interaction with customers, partners and suppliers through Web-based applications. The company Web site, besides processing orders, has become a customer service center providing technical assistance, software, and discussion forums. For registered suppliers it gives access to inventory and order fulfillment systems, as well as to the production schedules that are generated from order processing. The Web site is connected not to a single ERP system, but to a corporate intranet giving employees access to all business functions through Web-based applications. The intranet extends onto an extranet that supports Web-based EDI exchanges with partners and suppliers.

Cisco's structure is no longer that of a traditional company. Instead, it is network of business partners - a *virtual organization* [Sotto, 1997] - powered by an aggressive outsourcing strategy and tight integration of information systems. If outsourcing was a consequence of process reengineering, then the Internet and its associated technologies are paving the way to enterprise *re-organization*: the arrangement of business networks comprising the core competencies of several partners and supported by Internet-based integration technologies. Thus, new enterprise paradigms are arising such as the *virtual enterprise*, defined as [Camarinha-Matos and Afsarmanesh, 1999c]:

> "A virtual enterprise is a temporary alliance of enterprises that come together to share skills or core competencies and resources in order to better respond to business opportunities, and whose cooperation is supported by computer networks."

The virtual enterprise is a consortium of business partners that join competencies in order to take advantage of a certain market opportunity, while that opportunity exists. Its structure reflects the need for enterprises to quickly adapt to changing market conditions, with information technology as a facilitator of interaction and coordination. The challenge is no longer reengineering internal

[27] http://logistics.fedex.com/

processes but integrating and coordinating business processes from different partners. It is also not a matter of outsourcing non-core processes; the enterprise re-organizes itself and develops its business relationships into a network where partners share their competencies and resources in order to offer a product or service that meets market demands.

Besides Cisco, another example of such re-organization has been undertaken by PUMA AG [Mertens et al., 1998], the holder a traditional brand of sports articles. After a company crisis during the early 1990s, PUMA was radically restructured. PUMA has defined and concentrated on its core competencies: product development, design and marketing. Production was handed over to several Asian shoe manufacturers and most of the worldwide logistics carried out by a British transporter with a branch in Singapore. Product development is done in cooperation with international designer teams in Germany, USA and Taiwan. PUMA administers an extranet that connects customers, manufacturers, sales partners and licensees, a total of 180 independent partners embodying a single brand name.

Virtual enterprising may also lead to unforeseen opportunities as shown by the Brazilian VIRTEC project [Bremer et al., 1999]. VIRTEC[28] was a research initiative that was able to bring together ten small to medium-sized enterprises (SMEs) working in the vicinity of São Carlos. These had distinct know-how: electronics, mechanics, ceramics, polymeric materials, fluid systems, software, and exportation services. The purpose of VIRTEC was to increase competitiveness of SMEs and allow them to enter new markets by explore new opportunities, develop new competencies, and promoting the use of complementary services and resources.

Under VIRTEC, these independent companies would exchange market information in order to identify opportunities and select the appropriate partners for developing a new product. VIRTEC would then provide the necessary legal and technological infrastructures for these partners to create a virtual enterprise to produce and bring the new product to market. On a first approach, a matrix was developed having each enterprise represented in both a row and a column. For ten SMEs, this would be a 10x10 matrix. The diagonal would be empty but every other cell would describe the possibility of crossing competencies so as to create a new product. The cell in row i and column j would describe a set of potential products obtained by combining technology from enterprises i and j simultaneously[29].

Obviously not all combinations could lead to feasible potential products and some had to be completed with additional technology from other partners. Still, a year after its start, VIRTEC was supporting three virtual enterprises that had brought three new products to market. The most illustrative and successful one was a new hammer with a recyclable head made of polyurethane. This product, which has been sold in the US, resulted from the cooperation between a mechanic parts manufacturer, a polymeric manufacturer and the exportation services provider. Thus, in this case it was the virtual enterprise creating its own market, exploring an opportunity that perhaps not even the customers had foreseen.

As shown by these two examples, the product or service may be innovative if the virtual enterprise combines a unique set of competencies (as in VIRTEC), or it may be a regular product but delivered faster, at lower cost, or with improved quality (as in PUMA). This is possible because virtual enterprising, contrary to outsourcing, is not a one-way business relationship. The collaboration and sharing of resources in the virtual enterprise makes business partners aware of their respective roles, so they can operate with improved overall performance. Just as with process reengineering, where internal business units should integrate their tasks under a common process, enterprise re-organization

[28] http://www.virtec.com.br/

[29] This procedure was described by Prof. Bremer himself on the workshop "Cooperação em Redes de Empresas" held in Porto on 17th July, 2000.

into business networks requires partners to integrate and coordinate business processes across their boundaries.

2.5 Concluding remarks

The key features of the Internet having a strong impact on business practices can be summarized as follows [Timmers, 1999]:

- Availability: on the Internet, resources are available with immediate access 24 hours a day, 365 days a year, decoupling buyer from supplier, and supporting a global presence by overcoming time differences;
- Ubiquity: it is a global information network supported by fixed, mobile, cable and satellite communications worldwide and at low cost;
- Global: buyers access suppliers globally, suppliers get access to customers globally;
- Local: it reinforces local physical presence and enhances person-to-person business relationships as well as application interoperability;
- Digitization: it promotes the convergence of communications, information processing, event handling and media under a digital form;
- Multimedia: it allows a combination of technologies that can considerably enhance the promotion of goods and services;
- Interactivity: buyers and suppliers can exchange information in both directions;
- One-to-one: it is possible to profile customers and deliver information that is tailored to each customer's interests;
- Networking: it is possible to establish positive feedback loops that create increased benefit for increasing numbers of users, such as virtual communities, electronic auctions, and e-marketplaces;
- Integration: it is possible to deliver a wide variety of functionality through a single point of access.

Together, these features mean that the Internet and its associated technologies display an unparalleled potential to reshape the way business is conducted today. This chapter presented an overview of the origin and evolution of the Internet, and of the business practices that have emerged since then - collectively defined as *e-business*.

This chapter has also shown that major breakthroughs were not achieved through single developments, but usually from parallel trends of development that at some point converged, creating revolutionizing possibilities. It was Licklider's concept of "galactic network" developed in parallel with Kleinrock's packet switching theory that influenced Roberts in devising the ARPANET. It was from Cerf's experience in creating the NCP protocol and from Kahn's work on packet radio systems that the TCP protocol has resulted. It was Clark's implementation of TCP for personal computers together with Metcalfe's Ethernet technology that led to the widespread development of LANs. It

was Berners-Lee's work on hypertext together with the development of the HTTP protocol that have given rise to the World Wide Web.

Then a growing online research community interested in sharing information, together with business interests in exploring new marketing channels, would eventually lead to the first B2C e-commerce approaches. The development of wireless communications in conjunction with the Web's explosion would result in the Internet being accessible from mobile phones. The practice of EDI together with the development of Internet protocols and standards is giving new options for electronic data exchanges. The need for streamlining the supply chain (e.g. reducing inventories, accelerating time-to-market, sharing costs) together with connectivity and collaboration possibilities over the Internet promotes new kinds intimacy between business partners, and enables partners to integrate and automate their business processes.

Also, the development of comprehensive ERP systems in parallel with the implementation of Web front-ends for both customers, suppliers and any business partner in general, are originating Internet-aware enterprise information systems and making use of the Web browser as the point of entry to every functionality. Finally, trends in process reengineering have led enterprises to identify, analyze and improve their business processes, and companies have realized the need to focus on their core competencies; on the other hand, the integration and automation possibilities of the Internet have opened the doors for the enterprise to re-organize itself by taking advantage of networking capabilities in order to establish dynamic, possibly temporary, relationships with business partners, giving rise to organizational paradigms such as the virtual enterprise.

Following chapters will exhibit the same pattern, showing how different trends can be combined in order to bring significant benefits. This chapter has drawn attention to the combination of, on the technology side, the development of the Internet and its associated technologies and, on the business side, an increasing focus on core competencies. This focus is driving organizations to outsource non-core competencies and to dynamically establish relationships with business partners that provide those competencies. As each company focuses on its core competencies and relates with other companies that do the same, organizations evolve from all-encompassing enterprises into networks of competencies, as suggested in figure 2.1.

Although companies have since long been striving to become more efficient, future market conditions will require flexibility and adaptiveness. With faster and global access to information - about markets, products, partners, competitors - there are new market dynamics. Bringing the right product to market is just as important as bringing the product on time. Tomorrow, either competitors will be offering a better alternative or customers will be demanding something else. Being able to adapt and reconfigure the business structure to cope with these market dynamics seems to be the only way for being competitive. At least, that is what the examples in section 2.4 indicate.

Companies, thus, will realize the need for dynamic business relationships, instead of fixed business partnerships. The supply chain is a comprehensive and rigid structure, often incurring unnecessary costs or delays because planning is mostly based on forecasts, not on effective sales. This is known as the push method, where each tier pushes goods to the next tier, with retailers trying to sell to the customers everything that has come along the chain. Gradually, supply chains are adopting the pull method, where final customer sales drive the flow of orders all the way back to production. Initiatives such as VMI, CPFR and scan-based trading are attempts towards the pull method. Consequently, supply chains are sometimes referred to as supply/demand chains.

But with either push or pull methods, supply/demand chains still lack the flexibility required to cope with new business opportunities. Supply/demand chains assume that the product they are selling today will be the one being sold tomorrow. Increasingly demanding customers and increasingly global

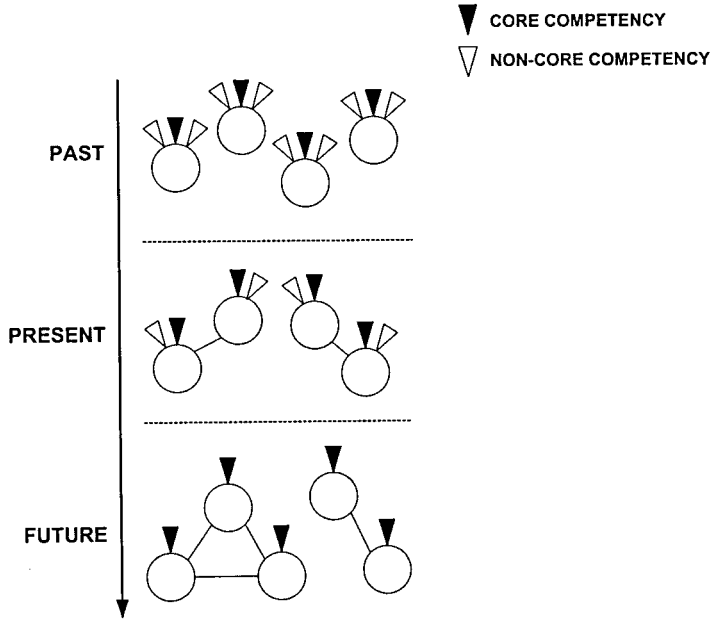


Figure 2.1: Evolution of the enterprise towards a network of competencies[30]

and aggressive competitors, however, will require improvements in products and customer services. Additional competencies will have to be inserted into the chain, while others may become obsolete. The chain becomes a network of competencies that aims at delivering an appropriate compound of product and service to the market. The supply/demand chain becomes a business network.

The Internet and its associated technologies are precisely the enablers for this new kind of organizations. The availability and accessibility of the Internet as a ubiquitous and inexpensive networking platform means that all the information, transactions and decisions will flow through that platform, or over platforms based on Internet technologies. These technologies support entirely new possibilities for searching, contracting, and performing business online.

However, taking advantage of the Internet as a business platform is not straightforward because a lot of functionality is still missing to support the integration of inter-enterprise business processes. Currently, only application integration has been successfully implemented on a large scale because it can benefit directly from existing Internet technologies. In order to integrate and automate business processes from different enterprises, new advances are required in cooperative work, business process modeling, simulation, workflow management, process monitoring and control, and information management. This work focus on improvements in these areas, particularly in applying workflow management to support the creation and operation of business networks.

[30]This figure was adapted from a conference presentation by Peter Weiß [Weiß and Kölmel, 2002]

Chapter 3

Workflow Management Systems

Research efforts that are relevant to business networking have been focusing more on configuration and organization issues, and less on coordination [Pontrandolfo and Okogbaa, 1999]. According to these authors, currently there is a need for a systematic approach to coordination that can be applied in networks of heterogeneous and dynamic resources. Whereas in the past coordination meant planning and controlling internal resources, business networking has extended the challenge of coordination to managing dynamic relationships between trading partners. Still, in a similar way to what has happened before, coordination means that enterprises must identify, improve and automate their business processes. But whereas once those processes extended across departments, now they extend across enterprise boundaries. Workflow management, which is a systematic approach to the coordination of business processes, and which has been typically employed in intra-enterprise environments, could prove to be useful in inter-enterprise scenarios as well.

The origin of workflow management can be tracked back to the 1970s with the first efforts towards automating office procedures [Zisman, 1977]. These efforts aimed at identifying and analyzing the routing of documents between employees and the processing each document has to undergo at each desk. Subsequently, this analysis would give a systematic view of how documents are transferred and processed through an office procedure. If these procedures are well known and documented, the purpose of each step can be specified to a greater detail, avoiding those situations where documents have to go back one or more steps in order to be corrected. Having a global view on the steps each document must go through allows the office to handle documents in an automated way because the sequence of tasks, as well as the purpose of each one of them, are clearly defined, and they can be repeated over and over again.

But in order to automate office procedures someone has to study them and then convey that knowledge to others. Therefore, methods to describe office procedures were required. Textual descriptions would be too cumbersome to elaborate and interpret, so graphical representations - called models - were used to give instant overviews of how the procedures were structured. These models were also used to suggest new ways for structuring those same procedures. Gradually, models were being used not only to describe but also to specify how procedures should be undertaken. Thus, there was a shift from procedures dictating models to models dictating how procedures should be performed. Devising models that both describe and shape procedures has become common practice and has evolved to become the foundation of workflow management: the use of models to drive the execution of task sequences.

During the 1980s, developments in information systems have further encouraged the automation of office procedures as growing software functionality would replace humans in several tasks with

benefits in time, accuracy, and reliability. At the same time, early information systems specifically targeted at office automation were also developed, based on two different approaches [Schulze and Böhm, 1996]: document-oriented and process-oriented systems, as shown in figure 3.1.

Document-oriented systems provided file-related functionalities such as archiving, indexing, searching, and retrieval. This kind of systems was the precursor of today's document imaging solutions [Saffady, 1993]. Document imaging solutions take advantage of the possibility of scanning any document format (such as paper, microfilm, slides, x-rays or CAD drawings) into electronic form, and provide indexing, archiving, searching and retrieval capabilities. Besides scanning documents, imaging solutions can handle files of virtually any type (text, spreadsheets, databases, graphics, images). They also allow the user to view files, establish hyperlinks or create annotations in files. Imaging solutions allow multiple users and are appropriate for collaborative environments.

Process-oriented systems gave more emphasis to the structure of office work and aimed at supporting well known procedures or at least predictable task sequences. Early systems had an implicit logic and were extremely rigid, with no possible deviation from that logic. Subsequent systems allowed for task sequences to be defined by the user and, through the development of process modeling languages and IT infrastructures, these systems have evolved into today's workflow management systems.

The purpose of this chapter is to present an overview of workflow management systems, including standardization efforts, commercial solutions and research projects in this field, so as to investigate the main characteristics and the kind of functionality that these systems usually provide. For this purpose, it is useful to start by defining workflow and presenting the Workflow Reference Model, although, it should be noted, the Workflow Reference Model is already a result of standardization efforts. Nevertheless, it has become the fundamental framework that reflects the currently accepted view on workflow management systems.

3.1 Defining workflow

Workflow is defined as (adapted from [WfMC, 1999b]):

> The automation of a business process, in whole or part, during which documents, information or tasks are passed from one resource to another for action, according to a set of procedural rules.

In this definition, a resource refers to a human or technological resource, that is able to perform one or more tasks within the context of a business process. Every business process is described by a process

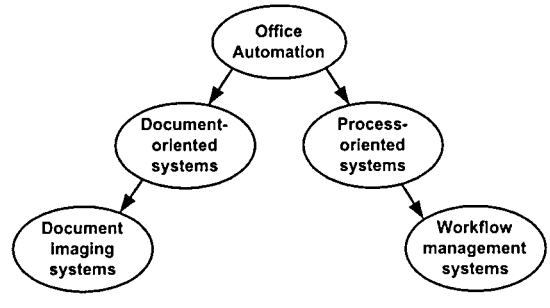


Figure 3.1: Evolution of office automation systems

definition that specifies the set of procedural rules. A workflow management system, also referred to as a workflow system, could therefore be defined as (adapted from [WfMC, 1999b]):

> A system that defines, creates and manages the execution of business processes through the use of software, which is able to interpret the process definition and interact with resources in order to execute that business process.

Workflow management exhibits the distinctive characteristic of separating *process logic* from *task logic* by defining two separate environments: one for the definition and enactment of processes, and another for the execution of individual tasks. The capabilities for defining and enacting processes are provided by the workflow management system, while resources are responsible for individual tasks. The two environments are related by control signals exchanged between the workflow management system and the resources: the workflow system asks resources to perform tasks, resources notify the workflow system on task completion.

The fundamental assumption of workflow management is that it is possible to analyze and model business processes. From that analysis, a process definition can be developed, either reflecting current or desired procedural rules. As illustrated in figure 3.2, a workflow enactment service will create process instances based on that definition and, by interacting with the assigned resources, it will control and track the progress of those instances. Therefore, workflow management comprises two environments: a built-time environment where processes are defined, and a run-time environment where those processes are brought into execution. Workflow management systems are thus obtained from a compound of functionality supporting process design and definition, process instantiation and control, and interaction with resources.

A task is an atomic unit of work to be performed by a particular resource, while an activity is a step on a business process that corresponds to the execution of that task. A process definition describes a business process usually by means of a sequence of activities assigned to particular resources. A process instance is said to be complete when all the activities have been fulfilled according to the

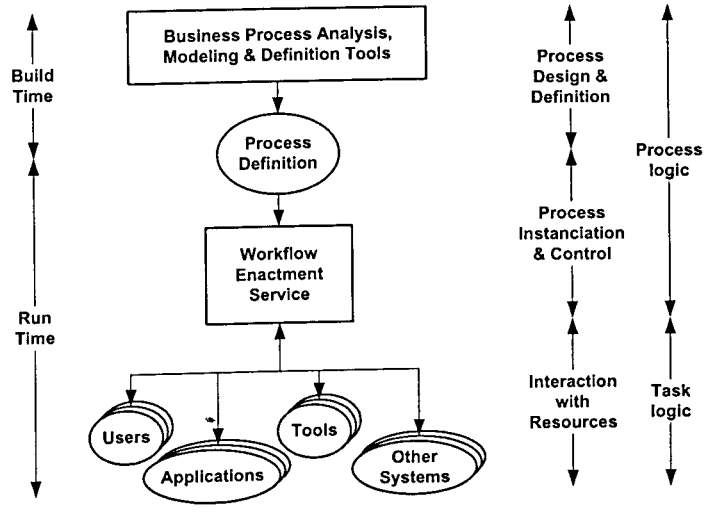


Figure 3.2: The framework for workflow management

process definition. Because activities are fulfilled by resources, a workflow management system must be able, for each activity, to interact with the corresponding resource. After that activity is complete, the workflow management system proceeds to the following ones, and so on, with each activity requiring the invocation of a particular resource.

A workflow management system thus requires integration facilities that allow resources to be invoked. It must be able to send to and receive information from those resources in order to fulfill each activity. In the early days of office automation, this meant distributing and coordinating tasks among a set of employees. Today, it means interacting with both employees and a variety of IT applications, tools, and systems found in modern business environments. Furthermore, the workflow management system itself can be considered as being a resource. This means that the challenge of integration is not only to provide the means for the workflow system to invoke particular resources, but to provide an integration infrastructure that allows all enterprise resources to interoperate; this challenge will be further discussed in the next chapter.

3.2 The Workflow Reference Model

The framework depicted in figure 3.2 shows that workflow management requires the ability to model and define business processes, the ability to execute those processes, the ability of resources to perform individual tasks, and the ability to interact with those resources. A workflow management system is thus a comprehensive system involving functionality that ranges from modeling business processes to controlling enterprise resources. In addition, workflow management systems should provide functionality to monitor the execution of business processes and should include the capability of interoperating with other workflow systems. In order to deal with this wide scope of functionality, workflow management systems can be seen as being comprised of several different components. Once these components are identified, the interfaces between them must be described.

The Workflow Reference Model [WfMC, 1995] describes these components and interfaces, i.e., it describes the general structure of a workflow management system. This reference model reflects the need to establish a common understanding of the purpose and scope of workflow management systems. The Workflow Reference Model, as illustrated in figure 3.3, identifies five types of components of a workflow management system: process definition tools that support business process modeling, workflow enactment services that support the execution of process definitions, the workflow client applications that interface a workflow enactment service with human resources (or participants), and the invoked applications that interface a workflow enactment service with other IT applications. The Workflow Reference Model specifies the interfaces between these components as well as an interface for interoperability between workflow enactment services (interface 4).

3.2.1 Process definition tools

The Workflow Reference Model does not impose specific process definition tools. Different methods may be used to model and document business processes and these tools may belong to the particular workflow system being used, or they may be supplied as a separate product. In any case, the outcome of business process modeling must be one or more process definitions that can be released for execution at the workflow enactment service. The Workflow Reference Model imposes a format that process definitions must follow in order to be exchanged between process definition tools and workflow enactment services. Interface 1 specifies this interchange format. The process definition,

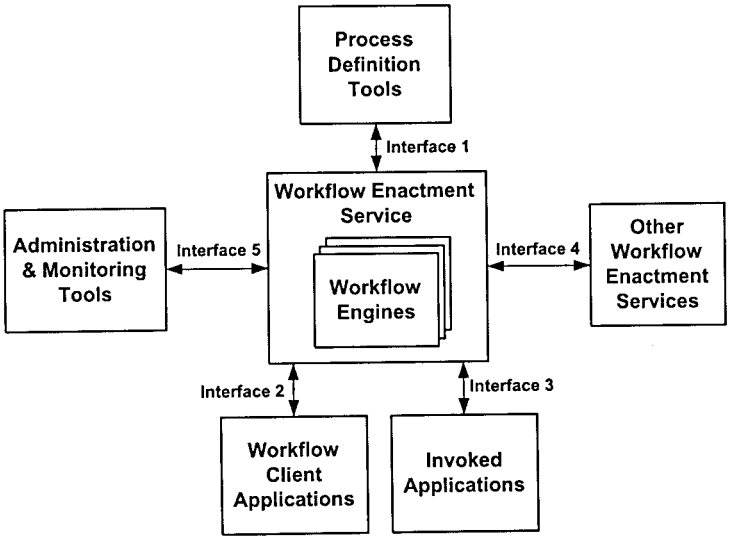


Figure 3.3: The Workflow Reference Model components and interfaces

expressed according to this interchange format, is the border between build-time and run-time environments (interface 1). The interchange format also allows a given process definition to be exported to other workflow management systems.

3.2.2 The workflow enactment service

The workflow enactment service creates process instances based on process definitions and enacts and controls those instances. The execution of process instances is attained by one or more workflow engines. The possibility of having more than one workflow engine may arise from the need to partition the process execution across the multiple engines. This partition may be done by process type, by functional distribution (each engine controlling activities assigned to certain resources) or by some other partition criterion. The presence of this possibility in the Workflow Reference Model comes from the intention to describe the general structure of any workflow management system, but does not mean that a workflow enactment service should have more than one workflow engine.

A workflow engine is responsible for most of the run-time features such as interpreting the process definition, creating and managing the state of process instances and of its constituent activities, navigating through activities sequentially or in parallel, dispatching tasks to resources, managing the flow of work items through the process, and implementing supervisory actions for administration purposes. The workflow engine can be considered as a state transition machine where individual process and activity instances change state in response to workflow events (such as the completion of a previous activity) or decisions within the workflow engine (such as branching through parallel activity paths).

At each point in time, a process instance may be in one of several states, as illustrated in figure 3.4. Upon creation, the process instance is inactive and is waiting the first activity to start. Once appropriate start conditions have been met, the first activities are triggered, leading the process into the running state. From this state, the process instance may be suspended and resumed, or may be successfully completed. It may also be abruptly terminated while running or being suspended. Each

individual activity in the process instance exhibits similar state transitions except for the termination state, since the execution of an activity is dependent upon the interaction with a resource. An activity may be considered as an atomic unit of work which is either executed entirely or rolled back to a restart point. This idea has led some authors to compare or relate workflow activities to database transactions [Sheth and Rusinkiewicz, 1993], which share that same behavior.

The Workflow Reference Model assumes that a workflow engine manipulates two kinds of data: *workflow control data* and *workflow relevant data*. Workflow control data are internal data that the workflow engine maintains concerning the control of process and activity instances. These data are for internal use only and not to be exchanged with other components. Workflow relevant data are used by the workflow management system to evaluate particular transition conditions that may affect the choice of the next activity to be executed. Such data are accessible to invoked applications and may be transferred and re-computed between activities. A third kind of data, *workflow application data*, is relevant only to the applications or user tasks and is not read or modified by the workflow engine, although the workflow engine may transfer it between activities.

3.2.3 Workflow client applications and invoked applications

A distinctive characteristic of workflow management systems is their ability to interact with human resources. Because workflow management systems had their origin in office automation systems, they assume that tasks assigned to human resources are piled on a queue of tasks until they are performed, just a employees used to pile stacks of papers on top of their desks. The interface of the workflow management system with a user is a task list or worklist displayed at the user's desktop, from where the user picks and performs the tasks he or she has been assigned and related work items such as documents. The task list is considered to be a workflow client application, hence the distinction between workflow client applications (interface 2) and other applications that do not require human interaction and can be invoked directly (interface 3).

The task list displayed at the user's desktop is an application that may be supplied either with the workflow management system or independently. In some cases, the task list is integrated into a common desktop environment alongside features such as e-mail and work-in-progress folders. Each task list is accessed by the workflow engine to deliver tasks and work items, and accessed by the user while retrieving those work items for processing and notifying task completion. A protocol must be defined between the workflow engine and the task list so as to add tasks, remove completed tasks, and suspend tasks. The activation or display of work items is under the control of the workflow client application alone. A task list may contain items relating to several different active instances of a

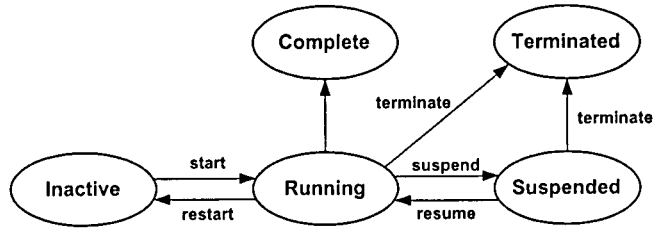


Figure 3.4: Example state transitions for a process instance

single process or from different process instances. The task list may also receive tasks from more than one workflow enactment service.

Besides workflow client applications, the Workflow Reference Model realizes that a workflow engine must be able to invoke any other potential application. This poses a challenge since the workflow engine cannot possibly cope with the logic of every application, and still has to be able to exchange workflow relevant with any application. The Workflow Reference Model circumvents this problem by requiring applications to implement a specific interface (interface 3). For those applications that are not compliant with this interface, application wrappers must be developed. The interface supports session establishment, the ability to start, suspend and resume activities, the notification on activity completion, and the exchange of workflow relevant data.

3.2.4 Interoperability with other workflow enactment services

The ultimate purpose of the Workflow Reference Model is to allow the interoperability between workflow management systems, or components of these systems, supplied by different vendors. Workflow enactment services and their constituent engines are the prime target for interoperability. One of the possible solutions is to make workflow engines exchange process definitions expressed in a common interchange format, the same interchange format used for transferring process definitions from modeling tools to workflow enactment services. But in many cases, processes do not run in isolation in each workflow engine. Processes running on different engines may have to be connected, nested or in some other way synchronized.

The Workflow Reference Model identifies four possible interoperability modes. The first mode is the chaining of business processes, i.e., the final activity of a process running on one engine is connected to the first activity of another process running on another engine. The second mode is the hierarchical relationship between business processes, i.e., the activity of a process running on one engine is actually a process that runs on another engine. The third mode is shared execution where one engine is responsible for some activities and another engine takes care of other activities belonging to the same process. The fourth interoperability mode is parallel synchronization: the execution of two processes running independently on different engines must be synchronized at some given step. The four interoperability modes are illustrated in figure 3.5.

In order to enable these interoperability modes, the Workflow Reference Model specifies interface 4 as an application programming interface (API). When an interface of the Workflow Reference Model is defined as an API, this API is referred to as a workflow application programming interface (WAPI). The WAPI specified by interface 4 allows one engine to request the execution of processes and activities running on another engine, and to remotely monitor and control the state of those processes or activities. This WAPI also supports the exchange of workflow relevant data between engines, as well as the exchange of process definitions. In some cases, the workflow engines may have different naming conventions for process definitions or workflow relevant data. In this case, the workflow engines are connected not directly, but through a gateway application that conciliates those naming conventions and interacts with each engine according to the WAPI.

3.2.5 Administration and monitoring tools

Within a workflow management system there will be, in general, a set of monitoring functions to log events and track the overall progress of process instances. These monitoring functions are intended to collect the necessary data to compute metrics such as performance measures or resource load. At

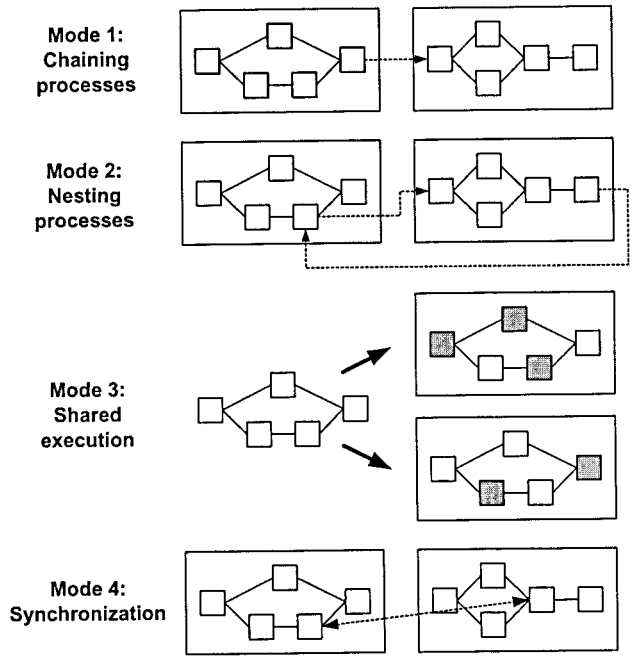


Figure 3.5: Interoperability modes between workflow engines

the same time, a set of administrative functions may be used to configure security, control and authorization constraints. The Workflow Reference Model does not restrict the scope of administration and monitoring tools, and allows them to implement other management functions such as acting as a repository and distributing process definitions. Interface 5 specifies the events and data structures for audit information that is collected by administration and monitoring tools. This interface does not specify how data should be stored, but specifies what information is to be gathered and presented for analysis.

3.3 Standardization efforts in workflow management

The scope and nature of workflow management systems is such that they must interact with many enterprise resources, while exchanging and managing an arbitrarily high volume of information from different sources. This challenge, if not approached properly, can lead to monolithic solutions that dominate the enterprise system architecture, that can hardly tolerate the existence of other management systems, and that are difficult to maintain and upgrade. In the beginning of the 1990s, different vendors developing workflow products independently led to a scattered landscape of non-interoperable workflow solutions offering contrasting, restrictive and inflexible functionality [Sheth et al., 1999].

The introduction of the Workflow Reference Model in the mid-1990s has brought some order to this landscape by providing a framework for relating workflow management systems and their capabilities. Since its introduction, the Workflow Management Coalition has been proposing several standards covering all the components and interfaces of the Workflow Reference Model. These standards have been seldom employed in workflow management systems. Nevertheless, they characterize

the challenges that workflow management systems must face, and describe features that those systems implement in one way or another.

In parallel with the efforts of the WfMC, other standardization bodies such as the Object Management Group (OMG) and the Internet Engineering Task Force (IETF) have brought contributions to the development of interoperable workflow management systems. These contributions reflect the particular perceptions and expertise of those standardization bodies. On one hand they tend to favor particular technologies such as distributed object systems or HTTP extensions, respectively. On the other hand, these contributions complement the standards from the WfMC because they bring insight into how workflow management systems can take advantage of emerging technologies.

3.3.1 The Workflow Management Coalition (WfMC)

The standards proposed by the Workflow Management Coalition (WfMC) are the foundation for what is known today as workflow management. The original purpose of these standards was to serve as interface specifications for developing interoperable workflow management systems and components. The approach and structure of the WfMC's standardization efforts is based on the Workflow Reference Model, as depicted in figure 3.3. After identifying the general components of a workflow management system, the WfMC focused on specifying in detail the interfaces between these components.

3.3.1.1 Interface 1

As previously described, the Workflow Reference Model requires the interface between process definition tools and the workflow management, interface 1, to be specified as a process interchange format. The WfMC has therefore proposed the Workflow Process Definition Language (WPDL) [WfMC, 1999a]. The WPDL is a formal, textual language that uses a specific grammar to describe business processes. When expressed in WPDL, a process definition is structured according to sections and subsections within these sections. These sections are populated with name-value pairs, most of which are standardized, although the WPDL specification allows the user to define additional name-value pairs. The structure of a WPDL process definition is illustrated in figure 3.6.

Every WDPL process definition contains a header stating general data such as the WPDL version being used, the product used to create the file, the date of its creation, an appropriate name and description, and the author name. The process definition then includes the first of its main sections, a list of workflow participants, designating the users, human roles, or organizational units that are assigned to activities of this process. The following section contains the list of applications to be invoked during the execution of those activities. For each application, its input and output parameters are named. The data section specifies the type for each of these parameters. The workflow section enumerates the individual activities and transitions. Each activity section indicates one application to be invoked (implementation element) and the assigned participant (performer element). Each transition connects one activity to another and may optionally include a boolean condition evaluating to true if the transition should be made or false otherwise.

Because a WPDL file is just a structured text file with sections and name-value pairs, it is particularly fit for being expressed according to an XML syntax. Expressing process definitions in XML would ease their use and manipulation since tools for parsing XML files are widely available, in contrast with WPDL. It would also bring several advantages since many applications today use XML as their preferred data format. Realizing these benefits, and considering that XML is being increasingly used in virtually all applications, the WfMC began working on an XML-based process definition lan-

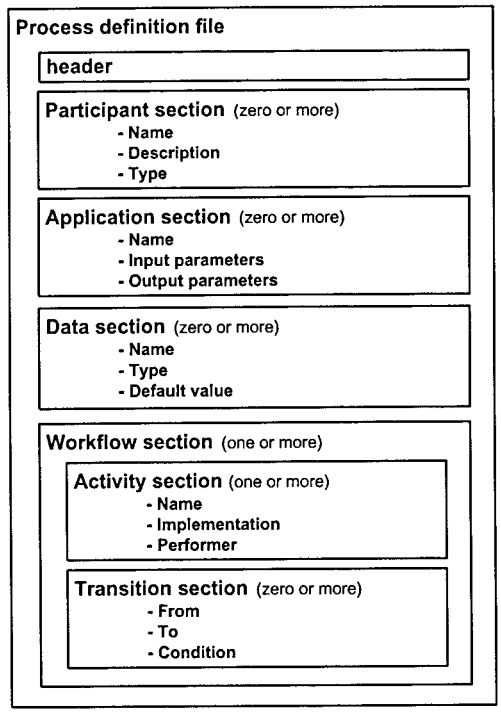


Figure 3.6: Structure of a WPDL process definition file

guage called XPDL [WfMC, 2001]. Apart from minor adjustments to its syntax, XPDL is similar to WPDL in every other respect and has exactly the same structure as depicted in figure 3.6.

3.3.1.2 Interface 2

The WfMC started by defining interface 2, the interface between the workflow enactment service and the workflow client applications, as an application programming interface [WfMC, 1996]. The purpose of this interface is to allow the development of front-end applications which need access to workflow enactment functions. As such, interface 2 is specified as a WAPI comprising a set of function calls that every workflow enactment service should implement. Hence the name of workflow client applications: the front-end applications are client applications of the workflow enactment service, which is functionally the server. Interface 2 also allows client applications to be developed independently while ensuring that they will work with any particular workflow product, as long as that product implements the specified WAPI.

Interface 2 specifies several groups of function calls that a workflow enactment service must implement.

- Connection functions delimit the interaction between a client application and the workflow enactment service. By issuing a connect command, the workflow client application informs the workflow enactment service that other commands will be originating from the application, until a disconnect command is issued. If the workflow enactment service is remote, these functions establish and terminate a communication session with the client application.

- Process control functions are those which change the execution state of one or more process instances. These functions allow the client application to retrieve any process definition, to enable or disable it for execution, to create a process instance from that definition, and to change the state of a process instance being executed. They also allow a client application to retrieve all the workflow control data and workflow relevant data concerning a particular process instance, to retrieve individual attributes from those data, and to manipulate those attributes. Because these functions may affect several resources, its usage is usually restricted to applications with certain privileges.

- Activity control functions are those which change the execution state of a particular activity instance. These functions allow the application to determine the possible states for that activity and to change that state. They also provide access to the workflow relevant data for that activity and allow to change or update the value of attributes from those data.

- Process status functions and activity status functions are intended to provide a view on the work done, the work still to be done, and the work load of participants or groups of participants. These functions allow the client application to request the list of process and activity instances under execution and to retrieve the record of completed work and of the work-in-progress for any of those instances.

- Worklist functions allow workflow participants to access information about work they have been assigned. During the execution of a process instance, any given participant may be assigned one or more activities. The participant sees each activity assigned as a work item and the collection of all work items assigned to that participant is the participant's worklist. The worklist functions allow the participant to retrieve the assigned work items and to complete them or to reassign them. They also allow the participant to retrieve and manipulate attribute values from the data of each work item.

- Administration functions allow client applications to take administration and maintenance actions on a workflow enactment service. They include functions to change the execution state of process and activity instances, functions to terminate and abort process instances and functions to assign attribute values to process and activity instances. Some of these functions are similar to previous ones. For example, process control functions also allow to change the execution state of a process instance, and to manipulate attribute values. The difference is that the administration functions can affect several process instances with a single function call.

3.3.1.3 Interface 3

After having defined interface 2, the WfMC has been revising it to include more functions. The WfMC realized that workflow client applications may need to invoke other applications. Conversely, invoked applications illustrated in figure 3.3 may also require access to workflow enactment service functionality. The WfMC has therefore included application invocation capabilities in interface 2 and the result was a joint specification of interfaces 2 and 3 [WfMC, 1998b]. The specification of interface 3 builds on interface 2 and introduces the additional functions for invoking applications.

- Application invocation functions allow to start and stop applications, exchange workflow and application relevant data with applications, and control their run-time status. Within these functions there are connection functions for establishing the communications session or login

procedure with invoked applications. Invocation functions are able to activate a specific application, terminate it, and request its status while the application is running.

Even though the WfMC specifies these invocation functions, it cannot be expected that applications that are external to the workflow management system will implement them. Realizing that workflow management systems and client applications may need to invoke any application tool in general, the WfMC proposes the concept of *tool agents*, which are no more than application wrappers that implement the application invocation functions and work as a gateway to invoke the desired applications. Instead of referring to application tools directly, the workflow management system must refer to the existing application wrappers for those tools.

3.3.1.4 Interface 4

The WfMC has defined interface 4 as the interoperability interface between workflow engines. This interface allows workflow engines to share or synchronize the execution of their process instances. But because a workflow engine is responsible for the execution of an arbitrarily large number of process and activity instances, it cannot afford to wait for another engine to complete a particular request before proceeding with its own work. While a workflow engine is busy creating a process instance or retrieving workflow data, the other engine must keep paying attention to events and to the progression of its own process instances. As a consequence, the interoperability interface between two workflow engines cannot be specified as a set of blocking function calls.

The WfMC specifies interface 4 as a WAPI that requires every workflow enactment services both to implement standard functions and to be able to call notification functions implemented by other workflow enactment services. The principle of interface 4 is that for every request function issued by a workflow enactment service there is a notification function that notifies that service on the completion of the request. Interface 4 thus comprises a pair of functions for each desired action: one function issues the request for that action from one engine to the other and another function notifies the completion of that request in the reverse direction.

Interface 4 includes request functions for creating, starting, and terminating process instances. Each of these functions has a notification counterpart that notifies the invoking engine upon completion of those requests. Additionally, there are some request functions that have no notification counterpart, such as functions for retrieving the state and data attributes of a process instance. There are also notification functions that have no immediate originating request function, such as the notification on process instance completion. This notification is a consequence of a process instance being created and started, but is generated only when the workflow engine completes the execution of the relevant process instance.

Interface 4 assumes that every workflow engine creating a process instance will be interested in being notified of events within that instance, but the invoking engine may choose not to be notified,

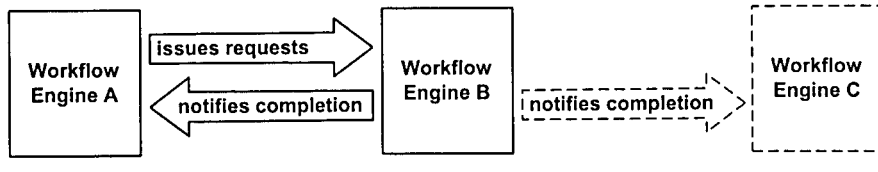


Figure 3.7: The principle for the WfMC's interface 4

either when it creates the instance by explicitly specifying that choice or at anytime during process execution by calling an appropriate function. Notification functions can be used to notify other workflow engines as well, besides the one issuing the request. For example, in figure 3.7 there could be a Workflow Engine C interested in being notified on the completion of a process instance that is running on Workflow Engine B but that was created and started by Workflow Engine A. This means that Workflow Engine B must provide a way for Workflow Engine C to subscribe to events from that process instance. The WAPI specified by interface 4 allows the situation of Workflow Engine C in figure 3.7 to occur, but does not specify how a third engine can subscribe events from a process instance it has not created.

The WfMC has prepared two technology bindings for interface 4. These are possible implementations of interface 4 using particular technologies. The first binding uses Internet e-mail with MIME encoding as the transport mechanism for messages exchanged between workflow engines [WfMC, 2000a]. Each possible message corresponds to a certain function call, so there is a one-to-one mapping between the function calls specified by interface 4 and the e-mail messages specified by this binding. For example, the subject field of the e-mail message indicates whether the message is a request message (equivalent to a request function) or a notification message (equivalent to a notification function). The message body contains the function name and any necessary function parameters.

The second binding uses XML documents to describe messages and HTTP as the transport mechanism [WfMC, 2000b]. As in the first binding, each XML message is equivalent to an interface 4 function call. But in this binding, special tags indicate whether the message is a request or a notification (the notification messages are called response messages). Additional tags indicate the function name and the values for its parameters. Special tags denote the execution state of the process instance, with one tag for each possible state. The XML binding proposes the HTTP protocol for communicating request and response messages between workflow engines.

3.3.1.5 Interface 5

According to the Workflow Reference Model, interface 5 is the interface between the workflow enactment service and the administration and monitoring tools. During workflow enactment, several events occur such as internal engine operations, WAPI events, and other events coming from external applications. The information that these events carry must be captured and recorded in order to be used later for analysis, status information, or performance measures. While there may be several technological solutions for the storage and retrieval of this information, the key issue is that the administration and monitoring tools must be able to interpret those information records so as to facilitate analysis. Even if these components are supplied from different vendors, an administration and monitoring tool from one vendor should be able to analyze the run-time information collected by a workflow enactment service from another vendor.

The WfMC has therefore developed specified interface 5 not as a WAPI, but as a data specification [WfMC, 1998a]. Interface 5 specifies what information is to be gathered and made available for analysis. This information is called Common Workflow Audit Data (CWAD) and it comprises a set of data records that describe the events that have occurred inside a workflow enactment service until a certain point in time. All the data records from the CWAD follow a similar structure, illustrated in figure 3.8. Each data record begins with a prefix that has the same structure for all records, then it includes a block of audit data that is dependent upon the specific event being recorded, and finally there is a suffix that contains optional workflow relevant data concerning the process or activity instance that was the source of the event.

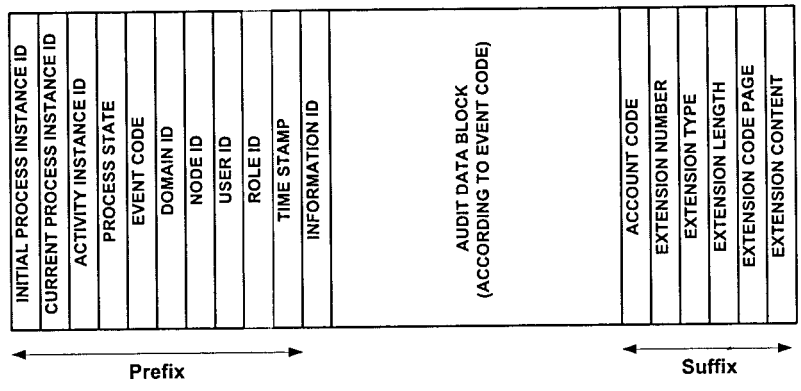


Figure 3.8: Structure of a CWAD data record

The CWAD prefix begins with an identifier of the process instance which is the source of the event, and a second process instance identifier in case the event was generated by a sub-process. The identifier of an activity instance follows, as well as the current state of the process instance. The event code denotes the type of event the record refers to. The domain, node user and role identifiers concern the primary user, whether a person or entity, involved with this event. The time stamp keeps the time at which the event was recorded. An information identifier indicates whether the audit data block contains standardized or proprietary data. This is because the WfMC realizes that workflow products may operate differently, and may need to record data other than those specified by interface 5.

Interface 5 specifies a set of standard event codes according to the operations that a workflow engine can perform. These operations generate audit information and interface 5 classifies this audit information according to different types:

- Process definition audit information records changes in process definitions. There is one event code for this kind of operations.
- Process instance audit information refers to events such as: a process instance was created or started, the state of a process instance has been changed, for example it may have been completed or terminated, or attributes of process relevant data have been changed. There is an event code for each of these operations.
- Activity instance audit information refers to the change in state of an activity instance, to changes in attributes of activity relevant data, and to the assignment or reassignment of individual work items. As before, there is an event code for each of these operations.
- Remote operation audit information is related to those operations performed on a workflow engine in reaction to the requests from a second, remote workflow engine. There are three event codes for each remote operation.

The remote operations are enabled by the interoperability WAPI (interface 4) and each request originates two events with two different event codes: one event is recorded on the source workflow engine when the request is sent, and another event is recorded on the target engine when the request is received. When the target workflow engine performs the requested operation, it records an event signaling that operation by means of another event code. Then the same happens with responses: one

event is recorded on the source (formerly the target) engine when the response is sent, and another event is recorded on the target (formerly the source) engine when the response arrives.

For remote operations, there is a different event code according to whether the operation is a request or response, whether it is being sent or received, and according to the particular operation being performed. The remote operation audit information thus possesses more event codes than the other types of audit information. And for each operation, three events are recorded: one when the request is received, one when the operation is being performed, and the third when the response is sent.

Regardless of referring to a local or remote operation, each event code dictates a different audit data block. Some examples are:

- the creation of a process instance originates an event with an audit data block of two fields: the identifier and the name of the process definition from which the process instance is being created;
- the change in state of a process instance requires an audit data block with other fields: the previous state of the instance and the new state for that instance;
- the change in state of an activity instance requires these and three additional fields: the identifier of the activity, the name of the activity, and the identifier of the application associated with that activity;
- the assignment of a work item gives an event with an audit data block of seven fields: the identifier of the activity instance, the identifier of the corresponding work item, the state of that work item, and then four more identifier fields (some of them optional) characterizing the user, node, role and domain assigned to the activity.

The suffix appended to a CWAD record contains a set of optional values which may be required by a particular workflow enactment service. The suffix is used in audit information from process instances and activity instances, and it is intended to convey workflow relevant data from those process or activity instances. These data are included as extensions to the basic structure of the CWAD record depicted in figure 3.8. If no extensions are included, the account code (the code of a work item) will be left blank and the number of extensions will be set to zero. Otherwise, the extension type and extension length fields will be used for each extension content.

3.3.2 The Object Management Group (OMG)

The WfMC standards specify the interfaces of the Workflow Reference Model regardless of the particular technologies that can be used to implement them. That is, in fact, the goal of the WfMC as it strives to reach consensus in the area of workflow management. In addition to these standards, the WfMC has presented technology bindings showing how some of those interfaces can be implemented using a particular technology. Because these bindings were provided for illustrative purposes, it is expected that software vendors and organizations will develop further technological approaches to the implementation of workflow standards. One such organization is the Object Management Group (OMG)[31].

[31] http://www.omg.org/

The OMG has proposed, within the framework of its Common Object Request Broker Architecture (CORBA)[32], a specification for implementing some of the workflow functionality described by the WfMC's standards. The specification is called the Workflow Management Facility Specification [OMG, 2000a], formerly known as the *jointFlow* standard [Schmidt, 1999]. The OMG specification focuses on the run-time infrastructure for the execution of process instances, and describes the implementation of that run-time infrastructure using CORBA technology. The Workflow Management Facility is based on the WfMC interfaces 2 (client applications), 4 (interoperability) and 5 (audit data). Although it does not follow exactly those interfaces, many features from the WfMC standards can be recognized in the Workflow Management Facility specification.

Because CORBA is an interoperability standard for software components, it can be used as the underlying middleware to integrate the Workflow Reference Model components. CORBA can be used as the communication mechanism between a workflow enactment service and the remaining components of the Workflow Reference Model: process definition tools (through interface 1), workflow client applications (through interface 2), invoked applications (through interface 3), other workflow enactment services (through interface 4), and administration and monitoring tools (through interface 5). Given that CORBA enables function calls across these interfaces, the Workflow Management Facility can focus on the implementation of the corresponding WAPIs.

But CORBA is also an object-oriented technology that favors object-oriented interfaces. For this reason, the software interfaces specified by the Workflow Management Facility differ slightly from those proposed by the WfMC, in that they are object-oriented. The Workflow Management Facility specifies a set of objects and their interfaces that a workflow enactment service should contain. The objects will encapsulate the implementation of the workflow enactment service functionality, while their interfaces will enable the other Workflow Reference Model components to communicate and interact with the enactment service. Figure 3.9 illustrates the class diagram for the Workflow Management Facility.

WfRequester The workflow requester or WfRequester is a class representing the request for some work to be done. WfRequester objects may be internal to the workflow enactment service or external. It may be an internal object that contains the trigger conditions for a given process and requests the execution of that process when those conditions are met. Or it can be an object implemented by an external application such as another workflow engine that is requesting the execution of a given process on this workflow enactment service. WfRequester represents the interface that an application must implement in order to be able to receive events from process instances that were created by that application.

The Workflow Management Facility defines two additional uses for the WfRequester: for linking processes and for nesting sub-processes inside other processes. In the first case, a WfRequester receives an event from a process that has just been completed and requests the execution of another process. In the second case, an activity from a running process plays the role of WfRequester: when the activity is started, it requests the execution of a sub-process; the activity then waits until the sub-process is complete and only then can the original process proceed to the following activities.

A feature worth noting is that one WfRequester may request the execution of several processes but for every single process there is only one WfRequester receiving the events that process generates.

[32] http://www.omg.org/corba/

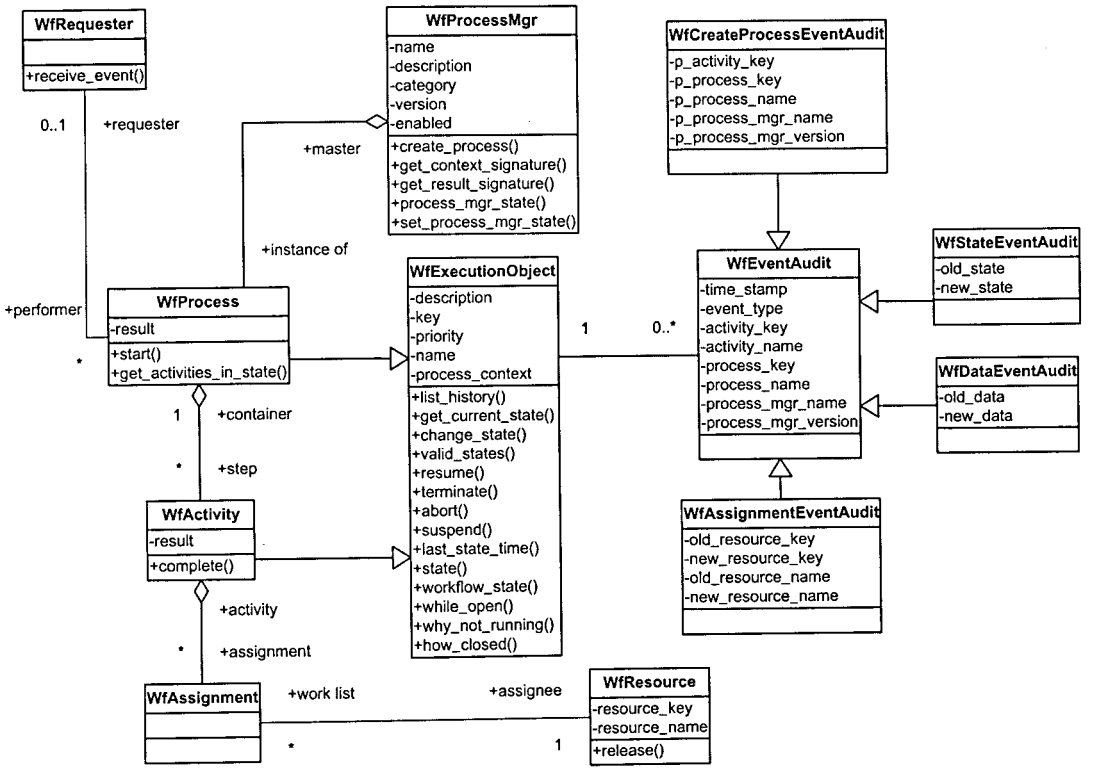


Figure 3.9: Class diagram for the OMG Workflow Management Facility

WfEventAudit A WfRequester receives all audit events from the process it triggers. Changes in process state, changes in activity state, changes in data or in assignment are all notified to the corresponding WfRequester. This is achieved by calling the WfRequester's method for receiving events, passing a WfEventAudit object as input parameter. The WfRequester is responsible for casting the WfEventAudit to the appropriate event subclass, namely WfCreateProcessEventAudit, WfStateEventAudit, WfDataEventAudit, or WfAssignmentEventAudit. These classes correspond to the main events defined by the WfMC's interface 5.

WfEventAudit thus represents audit records of workflow events. The WfEventAudit class defines a set of event attributes common to all workflow audit events: the time stamp, the event type, the process and the activity that was the source of the event. These attributes can be compared with the CWAD prefix defined by interface 5. The subclasses of WfEventAudit define additional attributes that are relevant to each event type: for example, WfStateEventAudit includes the old and new states for the source of the event, and WfAssignmentEventAudit refers to the old and new assigned resources. These attributes can be compared to the CWAD audit data block specified for each event code of interface 5.

WfExecutionObject The lifetime of a workflow event is not limited by the lifetime of its source: workflow events are persistent and can be accessed navigating the history of a process or activity. The event history describes the lifetime of processes and activities as they are created, proceed from state

to state, and are eventually complete or terminated. Thus, processes and activities have both an event history and a current state. The workflow execution object or WfExecutionObject is a base class that defines these and other common attributes of processes and activities. WfExecutionObject is never used directly as it is an abstract class. Its sole purpose is to ensure a common structure and behavior for process and activity instances.

A WfExecutionObject has methods to list the event history, which can be used by administration and monitoring tools for managing the retention, archiving, and deletion of workflow events. WfExecutionObject also defines common methods for changing the state of execution objects. This can be done by retrieving the set of possible states and changing the current state to one of those states, or it can be achieved directly by invoking a method for completing or terminating the execution of the object. The Workflow Management Facility assumes an execution object has state transitions similar to those of figure 3.4, but introduces the concept of states and sub-states depicted in figure 3.10. Although the Workflow Management Facility defines this state machine, it is possible to use and manipulate other states and transitions with the same WfExecutionObject interface.

WfProcessMgr The workflow process manager or WfProcessMgr is used to create process instances. It can be interpreted as the factory and locator for process instances. WfProcessMgr contains information about the input data a process requires (the input data is designated by the Workflow Management Facility as the *process context*) and about the results the process produces. Each WfProcessMgr object is identified by a name which is supposed to be unique within a certain business domain. Through the WfProcessMgr interface, it is possible to discover how many instances of a certain process are currently under execution, and to create additional ones. A WfProcessMgr object may be in one of two states - enabled or disabled - and it is able to create process instances only when it is enabled.

WfProcess A WfProcess represents a single process instance. When a process instance is created it enters the "not started" state shown in figure 3.10, and when it has successfully finished it enters the "completed" state. The context of a WfProcess (its input data) is set upon creation but before the process instance is explicitly started. The input data includes information about the individual activities, the assigned resources and the results the process should produce. This input data can be seen as workflow relevant data as defined by the WfMC.

A WfProcess contains zero or more WfActivity objects, each of which are, in general, assigned to a specific resource. The Workflow Management Facility introduces WfAssignment to decouple ac-

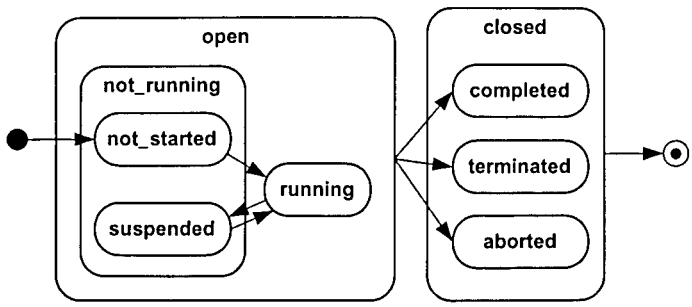


Figure 3.10: State transitions for a WfExecutionObject

tivities from resources and to express the possibility of a single resource having several assignments. This way it is possible to assign an activity to several resources, although the Workflow Management Facility leaves this issue up to the implementation of WfActivity objects. The Workflow Management Facility assumes that all assignments are created before the corresponding activity is ready for execution.

Every relationship in the class diagram of figure 3.9 is implemented by inserting a set of additional attributes or methods at both ends of the relationship. For example, the relationship of aggregation between a WfProcess and its constituent WfActivity objects imposes two requirements. The first requirement is that a WfActivity must have a reference to the containing WfProcess. The second requirement is that a WfProcess must include the necessary set of methods to determine how many activities it contains and to retrieve each of these activities. This way it is possible to retrieve, from a particular WfActivity, a reference to the containing WfProcess and to all other activities. Considering that WfProcessMgr also displays an aggregation relationship with WfProcess, it is possible to jump to other process instances and their activities as well. The Workflow Management Facility thus allows other Workflow Reference Model components to navigate through the objects within the workflow enactment service.

3.3.3 The Internet Engineering Task Force (IETF)

The growth of the Web and the maturing of Web-related technologies has brought new scenarios for employing workflow management. The Web excels in exchanging documents between resources, both humans and applications, and could easily become a major enabler of workflow management on a global scale. But the Web, based on short-lived HTTP connections, lacks proper support for managing the activities and processes that move information and documents around the network. Web protocols are able to transfer documents from one resource to the other, but are unaware of the purpose of each exchange and of the processing and routing that each document must follow. Web protocols do not support initiating computer- and human-executable activities, controlling those activities, or automating change notification [Bolcer and Kaiser, 1999]. Additional protocols are required in order to provide workflow features across the existing Web infrastructure.

A group of representatives from a handful of companies, which had already been working together in workflow standardization efforts within the WfMC and OMG, came up with an initiative for developing a protocol for workflow interoperability across the Web [Swenson, 1998a]. The goal was to develop an easy-to-implement protocol, rather than a comprehensive workflow architecture. The protocol should be kept light and simple, it should not reinvent anything unnecessarily, it should be extensible, and it should support long-lived workflow activities over the Web. Its purpose was to enable the interaction between resources which request work to be done and resources which perform individual tasks, all resources being identified by a Uniform Resource Identifier (URI). In particular, the protocol should be able to start, monitor, exchange data with, and control the work of a resource on a remote system. This initiative was submitted to the Internet Engineering Task Force (IETF) for standardization and the result was a draft proposing the Simple Workflow Access Protocol (SWAP) [Swenson, 1998b].

When related to the Workflow Reference Model, the SWAP protocol can be interpreted as a technology binding of interfaces 4 (interoperability) and 2 (workflow client applications) using the HTTP protocol. As far as SWAP is concerned, resources are distinguished by their Web address in the form of a URI. SWAP makes no assumptions about how these resources are implemented, and requires only that they return a proper and consistent result when requests are made to their URIs.

According to SWAP, there are different types of resources: some resources are responsible for creating process instances while others are assigned individual activities. The requests made to each of these resources depend on its type, as well as the expected results. SWAP defines each set of possible requests and responses as an *interface*. A resource that implements an interface is able to respond appropriately to the corresponding requests. A resource that implements more than one interface can handle more than one type of requests.

SWAP defines three fundamental interfaces that any resource may support. The ProcessDefinition interface is implemented by resources which are able to create process instances. For each business process there is a unique process definition URI. Sending a request to this URI is all that is needed to create a new instance of the business process. The second interface is the ProcessInstance interface, implemented by resources which maintain the state and manage the progression of a piece of work. The resource URI for a process instance can only be used from the moment of its creation to the moment of its conclusion. Between the two moments, a process instance undergoes a state behavior identical to that of figure 3.10. The third interface is the Observer interface that allows a process instance to communicate events to another resource. The URI of an observer resource can be given to a process instance. When the process instance changes state, it notifies the observer resource by sending it an appropriate request. There can be any number of observers for every process instance.

In order to complete a certain task, the process instance may need to invoke an external application or an asynchronous service. The SWAP protocol defines an additional interface, the ActivityObserver, that reports on external work that the process instance is waiting for. The ActivityObserver is to be implemented by process instance resources so as to provide the means for the process instance to describe what work is being performed. It is used by external resources as well to notify the completion a given task. The ActivityObserver interface is also a mechanism for nesting sub-processes inside other processes, because an ActivityObserver can provide a reference to the resource performing the work. If this resource is another process instance, then this one is effectively a child process of the first one. However, the ActivityObserver interface is most commonly used as an observer on the progress of individual activities.

SWAP proposes an implementation of these interfaces based on HTTP and XML. Requests and responses are encoded in XML with a set of tags according to the required parameters or results, respectively. These XML data are placed in the body of an HTTP message. Each HTTP message also includes standard HTTP headers but SWAP defines new values for the HTTP method header. In other words, SWAP defines new HTTP methods other than GET, POST or PUT. In fact, SWAP defines HTTP methods that correspond to interface methods. For example, CREATEPROCESSINSTANCE is a new HTTP method used for sending requests to a resource URI that implements the ProcessDefinition interface. The body of the request contains an XML document specifying the parameters for that request. In reaction to this request, the resource will create a new process instance and it will return an XML response with the relevant results. For responses, SWAP uses standard HTTP response headers, so the new methods are used only in requests. These methods have the same name as their corresponding interface methods.

ProcessDefinition interface The ProcessDefinition interface has three methods. The PROPFIND method retrieves the values for all attributes of the resource. It requires no parameters but returns several results, namely the interfaces the resource implements, a human-readable name and description for the resource, its globally unique URI and current state, a set of possible states, and information about the input and output data for the process instances created by this resource. The second method is CREATEPROCESSINSTANCE and it takes as parameters a URI to an observer, a human-

```
CREATEPROCESSINSTANCE /submit/order HTTP/1.1
Host: www.company.com
Content-Type: text/xml
Content-Length: ...
Authorization: ...

<swap>
     <observer>http://www.company.com/chair</observer>
     <name>equipment-purchase-process</name>
     <subject>procurement</subject>
     <description>New equipment purchase</description>
     <contextData>
          <z:length xmlns:z=http://equipment.com>200</z:length>
          <z:width>100</z:width>
          ...
     </contextData>
     <startImmediately>YES</startImmediately>
</swap>
```

Figure 3.11: Example of a SWAP request

readable name of the process instance to create, a subject and description, a collection of name-value input data pairs, and a flag specifying if the process instance should be started immediately or not. CREATEPROCESSINSTANCE returns a response containing the URI of the new process instance just created. The third method, LISTINSTANCES, allows the retrieval of the URIs for all process instances created.

ProcessInstance interface The ProcessInstance interface has five methods. It has a PROPFIND method similar to that of the ProcessDefinition interface but returning additional results concerning this process instance. The additional results are: the URI of the original ProcessDefinition, the list of activities (including URIs for the corresponding activity observers), an optional observer URI, a priority value, an optional URI to an user interface for this process instance, and a set of name-value pairs representing output data. Another method, PROPPATCH, is able to set any of these attributes simultaneously. The TERMINATE method cancels the execution of the process instance, accepting a text description of the reason for taking this action. Because there can be any number of observers for a process instance, the SUBSCRIBE and UNSUBSCRIBE methods manage a list of observer URIs that are notified whenever events occur in this process instance.

Observer interface The Observer interface has five methods as well. The PROPFIND and PROPPATCH methods manipulate a list of name-value pairs which represent relevant data from the process instance being observed. The COMPLETE method is used to indicate that the process instance has been completed and receives as parameter the output data from that process instance. The TERMINATED method indicates that the process instance is no longer running because another resource has requested its termination before it reached completion. A generic NOTIFY method is used by a process instance to notify events to subscribed observers. The NOTIFY method accepts a single parameter which is an event object containing several attributes: a time stamp, an event code, the old and new states in case it is a state change event, a set of name-value pairs in case it is a data change event and other optional attributes that are, in general, similar to those audit data attributes from WfMC's interface 5.

ActivityObserver interface The ActivityObserver interface has three methods. In a similar way to the previous interfaces, PROPFIND allows the retrieval of a number of attributes. In particular, an optional attribute contains the URI for a ProcessInstance if the activity has a sub-process associated with it. In addition, an ActivityObserver contains some unique attributes such as: a list of names of the assigned people, a URI to an user interface that can be accessed by a regular HTTP GET method, a deadline date and time, a list of name-value pairs representing input data for the activity, and a list of output data values. The PROPPATCH method allows the output data of the activity to be set. This method is used together with the COMPLETE method that notifies the completion of the activity. The process instance implementing the ActivityObserver can use the activity output data as workflow relevant data to determine and proceed with the execution of the following activities.

3.4 Commercial applications

Workflow management has found application in many areas such as insurance, banking, healthcare, public agencies and online services [Lawrence, 1997]. In several situations where specific task sequences must be enforced, workflow management is a helpful approach to control and monitor those processes. It is thus not surprising that ERP system vendors are devoting more attention to workflow management, with several of those systems already providing their own workflow functionality [Müller and Stolp, 1999]. At the same time, workflow is recognized as an important ingredient for enterprise application integration [Linthicum, 2000]. And workflow has been considered as one of the enabling technologies for Customer Relationship Management (CRM), together with the Web and data warehousing [Handen, 2000]. With special relevance for this work, workflow is considered to be the top layer of a reference architecture for interoperable e-business applications [Yang and Papazoglou, 2000].

Workflow management has been employed in many fields of business and research, and today there is a large number of workflow products and prototypes being used in various environments and for various purposes. Enumerating and comparing all available workflow solutions is an insurmountable task because there are simply too many products and not enough information on each one of them. Even if that could be done, it would be extremely difficult since there is a general tendency for each product to redefine basic workflow concepts and to employ an approach that is in some way different from every other. This explains the need and importance of standardization efforts such as those of the WfMC. Still, considering that any given workflow solution is as a compound of the elements shown in figure 3.3, workflow management systems can be compared according to the way they implement those interfaces and components.

The 1990s have witnessed an explosion of the number of workflow management systems available in the market. The number of workflow products reached a peak around 1996, when 200 to 300 workflow solutions were being proposed by different vendors [Sheth et al., 1999]. This large number of available workflow products can be explained by a diversity in features that these solutions offer. Each product is developed with a specific purpose or business environment in mind, and vendors often claim that their product has unique capabilities. In most cases, product features arise from different interpretations of what workflow management is and how it can be applied, but that does not necessarily lead to new capabilities.

The capabilities of most workflow products available today can be seen to fall within one or more of three major areas [Schulenburg, 1998]:

- Analysis, modeling, simulation and optimization (type A) - these applications place their main focus on the analysis and definition of business processes. Workflow management systems that are capable of simulating and optimizing process definitions are also included in this group. In some cases, the functionality of these applications overlaps with that from business process reengineering tools.

- Development and production operational applications (type B) - these include the workflow management systems that focus on the enactment and execution of business processes. Typically these systems comprise a workflow engine and the necessary application front-ends for users (the resources) to receive and perform their tasks. In some cases, these systems also provide development environments for building customized graphical user interfaces.

- Document management and office communication (type C) - these workflow management systems put the emphasis on the management of information including document storage, indexing, retrieval, communication and exchange between users. Although they provide the means for structuring and sequencing tasks, some of these systems are more focused on enabling communication rather than on modeling complex business processes. In some cases, these applications overlap in functionality with groupware and CSCW applications [Borghoff and Schlichter, 2000].

These application areas are strongly related to the three areas depicted earlier in figure 3.2 on page 65: (1) process design and definition, (2) process instantiation and control, and (3) interaction with resources. By developing expertise and building experience on each of these areas, software vendors have come up with workflow solutions that have as much in common as they have in contrast with each other [Sheth et al., 1999]. Figure 3.12 shows the result of a survey conducted in mid-2002 and enumerates some of the workflow management systems available by that time. The purpose of the survey was not to be exhaustive but to collect information about a significant number of workflow management systems, so as to characterize the spectrum of available solutions.

Workflow management systems dwell in a market that is as volatile as any other IT-related field. Constant innovation, increasing capabilities, growing competition and high customer expectations shape the dynamics of this market: products become obsolete and are eventually discontinued (e.g. FileNet's Visual Workflo), new products become available (e.g. HandySoft's BizFlow), some vendors reshape their products (e.g. IBM FlowMark turning into MQSeries Workflow), and some companies change while their products live on (e.g. Eastman Software becomes part of eiStream). In parallel with these changes, a few products where able to grow steadily (e.g. Ultimus).

Not all workflow products put the same emphasis on the three areas (types A, B or C). This is not to say that the unmarked areas mean that the corresponding products completely lack those functionalities. At the same time, two products in the same area are not necessarily functionally equivalent. Instead, the purpose of figure 3.12 is to show the strongest points of current workflow products. Most workflow products are clearly oriented towards enabling process-based operations (type B). Then a different line of products seems to put more emphasis on document management and office communication (type C). And only one in every four workflow management systems provides some functionality for business process analysis (type A).

The results of this survey also show that most workflow management systems available today make use of Internet-related technologies such as HTTP or e-mail. In fact, nowadays these are common approaches for delivering work items to human resources, in contrast with earlier times when workflow management systems where typically monolithic and employed proprietary mechanisms

Vendor	URL	Product	Type A	Type B	Type C	Compliance with Standards	Uses HTTP or e-mail
Abydos	http://www.abydosweb.co.uk/	ProcessControl		x			x
Accentuate Systems	http://www.accentuate.com/	SamePage			x		x
Action Technologies	http://www.actiontech.com/	Metro	x	x			x
A-Frame	http://www.a-frame.com/	WorkMovr		x			x
Apt Projects	http://www.aptprojects.com/	TCM System			x		
Ascent	http://www.ascenttechnology.co.uk/	Ascent Workflow		x			x
Axonet	http://www.axonetinc.com/	BizConductor		x	x		x
BancTec	http://www.plx.com/	Plexus FloWare		x		x	x
Callflow Software	http://www.samadhisoftware.com/	Syncro	x	x			
CCR	http://www.winocular.com/	WinOcular			x		
COI	http://www.coi.de/	BusinessFlow	x	x			x
Cyber Studio	http://www.cyberstudio.fr/	OpenCS		x	x		x
DST Systems	http://www.dstsystems.com/	Automated Work Distributor			x		x
Eastman Software	http://www.eastmansoftware.com/	Enterprise Workflow		x	x		x
eiStream	http://www.eistream.com/	ViewStar		x			x
ePaperlessOffice	http://www.cabinetng.net/	Cabinet NG			x		x
FileNet	http://www.filenet.com/	Panagon WorkFlo	x	x			x
Fujitsu	http://www.fujitsu.com/	i-Flow		x			x
Gauss	http://www.gaussvip.com/	VIP Enterprise		x	x		x
HandySoft	http://www.handysoft.com/	BizFlow		x			x
Holosofx	http://www.holosofx.com/	BPM	x	x		x	
HP	http://www.hp.com/	Process Manager		x			x
IBM	http://www.ibm.com	MQSeries Workflow		x		x	x
Identitech	http://www.identitech.com/	FYI FloWorks		x			x
Kaisha-Tec	http://www.activemodeler.com/	ActiveModeler/ActiveFlow	x	x		x	x
Lane Telecom	http://www.lanetelecom.com/	PASSPORT			x		
Lexign	http://www.icomxpress.com/	Lexign Flow		x	x		x
Ley	http://www.ley.de/	COSA Workflow	x	x		x	
Macess	http://www.macess.com/	I-MAX		x	x		
Meta Software	http://www.metasoftware.com/	Workflow Analyzer	x			x	
Metastorm	http://www.metastorm.com/	e-Work		x	x		x
mindwrap	http://www.blueridge.com/	Optix		x	x		x
Optical Image Technology	http://www.opticaltech.com/	OptiFLOW		x	x		x
Optika	http://www.optika.com/	Acorde Process		x			x
Orbisoft	http://www.orbisoft.com/	TaskManager		x			
ParaRede	http://www.pararede.pt/	OfficeWorks			x	x	
Pavone	http://www.pavone.de/	Espresso Workflow	x	x			
Promatis	http://www.promatis.com/	INCOME	x	x			
Q-Link Technologies	http://www.qlinktech.com/	Q-Link		x			x
Quality New York	http://www.qualitynewyork.com/	Paradigm Quality		x			
Same-Page	http://www.same-page.com/	eStudio			x		x
Savvion	http://www.savvion.com/	BusinessManager		x			x
Singularity	http://www.singularity.co.uk/	ActiveWorkflow	x	x			
SISED srl	http://www.sised.it/	Archind			x		
SolCom	http://www.solcominc.com/	SolCom		x			x
Sparta Systems	http://www.sparta-systems.com/	TrackWise		x			x
Staffware PLC	http://www.staffware.com/	Staffware		x		x	x
Teamware	http://www.teamware.net/	ProcessWise/Flow	x	x		x	x
TIBCO	http://www.tibco.com/	InConcert		x			
Tidemark Computer Systems	http://www.tidemarksys.com/	Advantage		x	x		x
Tower Technology	http://www.towertechnology.com/	Tower IDM		x	x	x	x
Ultimus	http://www.ultimus.com/	Ultimus Workflow Suite	x	x		x	x
W4	http://www.w4.fr/	W4 Workflow		x			x
WorkDynamics Technologies	http://www.work-dynamics.com/	ccmMercury		x			x
Worktiviti	http://www.udms.com/	Flotiva		x			x

Figure 3.12: Survey of workflow vendors and products (June 2002)

for interacting with users. Despite the increasing use of widely available technologies, workflow management systems were not equally committed to endorse workflow standards. It is somewhat striking that the majority of products does not refer to workflow standards and it should be noted that, from those which do, there is not a single product supporting all the WfMC interfaces. The seventh column in figure 3.12 marks products that implement at least one interface as defined by the WfMC.

Having demonstrated support for interfaces 1, 2, 3 and 4, Staffware seems to be in the leading position concerning compliance with standards.

Because workflow products hardly follow standards, the market for workflow management systems is a scattered landscape of solutions that propose a myriad of different approaches to the management of business processes. Workflow products differ in the scope of business process they support, in the way they model these processes, in their capabilities to interact with resources, and in the way they perform those interactions. Workflow products offer contrasting functionality. If this functionality is mapped against the Workflow Reference Model, then it can be seen that these products differ in the way they implement its components and interfaces. Thus, workflow products share the same underlying rationale. The downside is that, because there is no set of commonly agreed design principles, workflow products are, in general, not interoperable.

The results of the survey further suggest that, although there are three functionality areas, there are mainly two threads of development in workflow management systems. Figure 3.12 lists 55 workflow products, 45 of which are focused on the enactment of business processes (type B). From all the 13 products providing functionality for the analysis of business processes (type A), in 12 of them this functionality is actually an add-on to a product of type B. But when considering the 20 products with document management and office communication capabilities, only 11 of them are type B products. This suggests that there are two main kinds of workflow management systems:

- those which start from business process modeling and enactment, and subsequently develop the necessary infrastructure in order to interact with resources (top-down approach according to figure 3.2);
- and those which focus first on that infrastructure, and only afterwards consider functionality for modeling and enacting the business processes they support (bottom-up approach according to figure 3.2).

The following paragraphs describe one representative product from each of these two threads of development: for the top-down approach, Staffware, which is a mandatory reference in the workflow management arena, and for the bottom-up approach, OfficeWorks, a workflow management system that began as a solution for office communication.

3.4.1 Staffware

The product known as Staffware [Staffware, 1999] is perhaps the most illustrative (and successful) workflow management system from those available in the market. Its architecture and functionality span a wide range of issues and features that make Staffware one of the most comprehensive workflow products. There are few workflow products that can compete with Staffware and in most cases its functionality can only be obtained with a compound several other products. From this point of view, Staffware is unmatched because it might be more complete than any other product. On the other hand, however, it is quite a representative product because it has a typical architecture and the whole of its functionality has been designed and implemented in ways that are similar to many (if not most) workflow products.

Staffware is able to automate a wide range of business processes found in many organizations. According to the terminology of Staffware, a business process is a general concept comprising one or more workflow *procedures*. A procedure is composed of a number of distinct steps which act as a container for data and links to other steps. An instance of a procedure is referred to as a *case*. Each

case contains a number of tasks assigned to individual users. Users receive these work items via a *work queue*. Put simply, Staffware refers to business processes as *procedures*, it refers to process instances as *cases*, and it refers to task lists as *work queues*. Staffware provides tools for defining business processes (procedures), for creating and running process instances (cases), and for managing the tasks assigned to each user (work queues). Staffware provides other tools such as reporting and, more importantly, it provides the means to extend its functionality in several ways.

3.4.1.1 Client/server technology

One of the key features of Staffware is its ability to interoperate with several client/server technologies. In fact, Staffware is a client/server system itself. On the server side there is the Staffware Workflow Server. On the client side, graphical client applications run on user desktops and communicate with the Staffware Workflow Server through the local area network. For example, the Graphical Workflow Designer (GWD), which is a process editor, is a client application that supports the definition of procedures to be run on the server. The Staffware Workqueue Manager is another client application that provides users with access to their work items. In addition to client applications, Staffware also connects with existing enterprise or legacy systems. All connections from the Staffware Workflow Server to client applications and legacy systems is achieved by means of Remote Procedure Call (RPC) mechanisms and TCP/IP sockets.

Staffware can also be used in a multi-node setting with more than one Staffware Workflow Service. The existence of several server nodes improves fault tolerance and workload distribution. In other cases it may be required when geographically remote sites need to participate in a common procedure. For example, a procedure can be spread across two servers. Staffware requires one server - the master node - to be the owner of the procedure and that is the only node where the procedure definition can be changed. Other servers are slave nodes and receive their work from the master node. Master nodes and slave nodes exchange administrative information and synchronize that information when changes occur. One node can be a master for some procedures and a slave for other procedures. On a local area network, two server nodes may communicate by TCP/IP and over a wide area network it is possible for the servers to exchange data using e-mail.

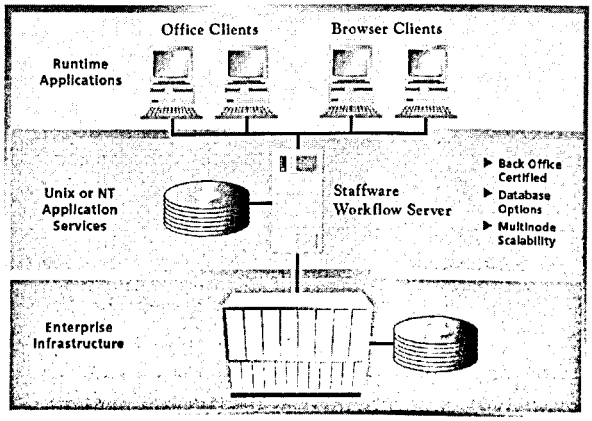


Figure 3.13: Staffware client/server architecture[33]

[33] Reprinted with permission from Staffware PLC

3.4.1.2 Staffware clients

Client applications are integrated with the Staffware Workflow Server by means of a layered software architecture. Staffware already provides client applications for most purposes such as defining procedures, managing work queues, and reporting audit trails. These are called *out-of-the-box clients* because they are supplied with Staffware. For tasks that require a graphical user interface (GUI), it is possible to build the form that is suitable for each task. There are several possibilities for building these forms: one is the Graphical Form Designer (GFD) supplied with Staffware; another is using Visual Basic for Applications (VBA), a Microsoft technology that allows users to build forms that look like applications they already know; a third possibility (not depicted in figure 3.14) is Open Forms, a Staffware mechanism that allows users to invoke any application as the GUI for a given task.

Besides the out-of-the-box clients, the layered software architecture encourages the development of new functionality by means of Staffware Enterprise Objects (SEO). SEO is a multi-language, object-oriented programming interface that provides access to all Staffware workflow functionality. With SEO, users can develop their own client applications tailored to specific needs of the enterprise; hence, these clients are known as *enterprise clients*. In addition to SEO, Staffware provides application gateways for Microsoft Exchange and Lotus Notes. This way it is possible to use third-party communication packages to exchange work items between the Staffware Workflow Server and the users. These are referred to as *external clients*.

3.4.1.3 The Staffware Workflow Server

The inner structure of the Staffware Workflow Server comprises several layers as shown in figure 3.15:

- The Communication Layer enables TCP/IP communication with client applications.
- The File Interface Layer (FIL) provides all components, including clients, transparent access to data that reside on a file or database system.

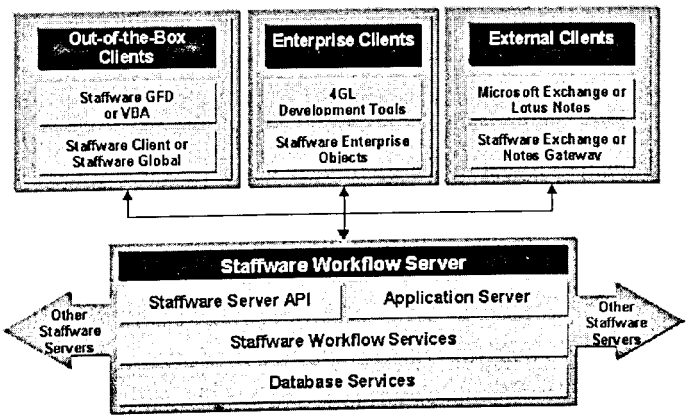


Figure 3.14: Types of Staffware client applications[34]

[34]Reprinted with permission from Staffware PLC

- The SEO Application Server manages the connections between the server and SEO-based client applications.
- The Staffware Application Layer (SAL) is responsible for client services such as logging in and logging out, opening and releasing work items.
- The Workqueue Services supply users with information about queues they have access to, and about work items in those queues.
- The Workflow Services are the heart of the Staffware Workflow Server since they interpret procedure definitions, create cases, route work items to work queues, and invoke external applications.
- The Database Layer Services give access to a database system that stores all workflow data. The schema for this database is publicly available so that other software vendors can develop additional reporting tools based on Staffware data.

The Staffware Workflow Server needs to store and update a significant amount of information such as case control data, case relevant data, audit trail data, work queues and user data. For this purpose, Staffware requires a relational database management system. It further requires that database system to be one of three possibilities: either Microsoft SQL Server, Oracle, or Staffware Classic. Apart from these particular products, this requirement resembles a common trend in workflow management systems: the fact that workflow management systems often require infrastructural support from complementary services such as messaging or database management. These requirements are not always evident from product descriptions.

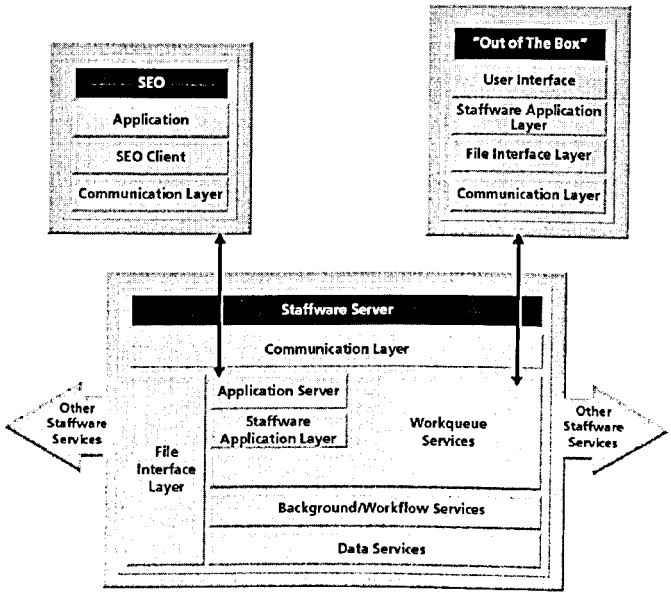


Figure 3.15: The Staffware Workflow Server software architecture[35]

[35]Reprinted with permission from Staffware PLC

Staffware has another infrastructural requirement that is less obvious. The communication between the server and client applications, and between the server and the legacy system can be achieved by means of TCP/IP, X.400 and other protocols. Staffware assumes that all resources of the enterprise IT infrastructure can be reached with these protocols. Because several types of hardware systems, software systems, and local networks may coexist in the enterprise, Staffware assumes that there is an enterprise-wide backbone connecting all these resources. It should be noted that Staffware does not provide this backbone, although it is mandatory for its operation. Many other workflow management systems have similar infrastructural requirements, and some even require specific third-party communication or messaging products in order to operate.

3.4.1.4 Process modeling

The Graphical Workflow Definer (GWD) is an out-of-the-box client application for defining procedures graphically. It is aimed at a non-technical audience who understand their business processes, but the simplicity of this tool is not a restriction to the complexity of procedures that can be defined. The GWD has a two-way link with the Staffware Workflow Server so that it can be used to create and maintain running procedures. The GWD is able to import process models from some third-party

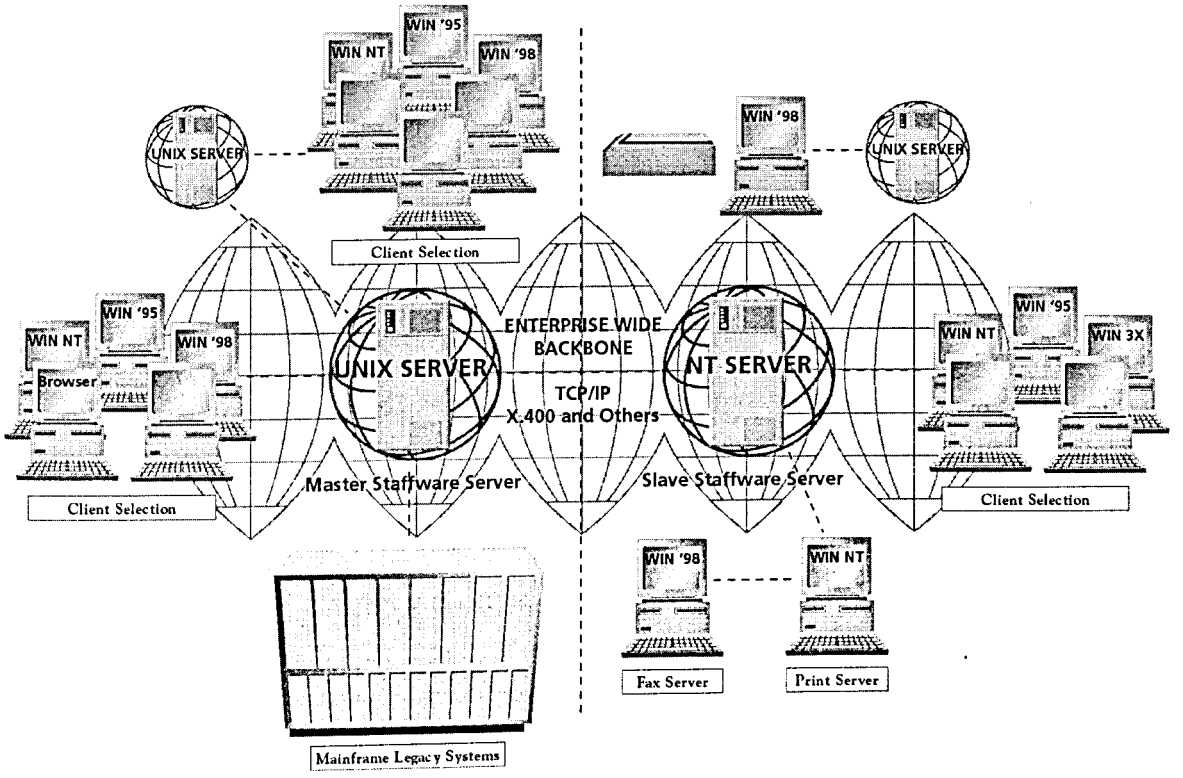


Figure 3.16: The Staffware physical system architecture[36]

[36]Reprinted with permission from Staffware PLC

BPR and simulation tools, enabling the analysis of business processes with external tools that provide additional capabilities.

The purpose of the GWD is to ease the definition of procedures by first specifying the procedure steps and then, for each step, by defining its associated data requirements and user involvement. The GWD proposes a modeling language with several graphical constructs. These constructs are used to describe four main types of steps:

- Normal step - represents an interaction with the user, and can be achieved in a variety of ways using any kind of client application (out-of-the-box, SEO, or external). It may need no special-purpose user interface if it is used simply for notifying the user of some event. A normal step represents a task that will be added to one or more work queues.

- Automatic step - enables the automation of the activity related to a given step. Automatic steps are used to invoke external applications that run without user intervention. For example, it may be used to exchange data between Staffware and external systems, or to invoke external programs, or even to print documents automatically. Automatic steps do not show up in work queues.

- Event step - it makes the case wait until some special conditions have occurred. Event steps are used to synchronize the step with an external event, effectively pausing the running case. An event step may be also used to synchronize two workflow procedures as shown in figure 3.5 on page 70, interoperability mode 4.

- Open Client Step - enables the integration of Staffware with external client applications, particularly Microsoft Exchange and Lotus Notes. The open client step is a combination of the automatic step and the event step: Staffware invokes the external client and suspends the step until the client application issues an event that tells it to proceed.

The GWD also allows existing procedures to be used as sub-procedures in other procedure definitions. The full procedure, including any sub-procedures, will have a consolidated audit trail. Besides these constructs, Staffware has additional mechanisms that can be used in procedure definitions:

- Deadline - associated with steps, a deadline is a modeling construct used for time-based alerting. This deadline may be defined at build time or computed dynamically during run-time, based on an expression. Once a deadline expires, another sequence of steps may be initiated.

- Routing, branching, conditions, parallelism and wait - these constructs are used to establish the routing logic of the procedure, and to react to events at run-time. A wait step can be used as a point of synchronization for multiple parallel routes in the procedure.

Staffware uses data fields as attributes for individual steps. These fields are workflow relevant data that can also be used in forms to be sent to the users. There are four types of fields: simple scalar, memo, attachment and composite. A simple scalar is a text value denoting a number, date, time, or currency. A memo is a text field that users can manipulate directly. An attachment is a physical file name to be associated with the step. A composite is a database table containing one or more fields. The values of these fields are sent to the users assigned to the corresponding step.

3.4.1.5 Users, groups and roles

Staffware users are defined within the Staffware Administration Manager. A new user must be defined for each person using Staffware. Each user has a unique login identifier and a password. As a consequence of creating a user, a new work queue will be created for that user. The Staffware Administration Manager is able to configure supervisors for each user, that have access to the same work queue for monitoring purposes. User groups can also be configured. Creating a work group automatically creates a work queue for that group, and existing users may be assigned to any group, which will then have access to those group's work queues. User roles are a third possibility that does not imply a new work queue. Rather, work is redirected to a specific user queue. A role has only one possible user, though a user may have many roles. Users, work groups and roles are all available to the GWD, which allows different kinds of assignment.

3.4.1.6 User interface for tasks

The Staffware Workqueue Manager is an out-of-the-box graphical interface for users to access their work queues. It allows users to open and release their work items, as well as forwarding a work item to another user. (The Staffware Workqueue Manager is also the client application used by supervisors of work queues.) Each work item, when opened, has its own user interface. This interface can be built according to three possibilities, as mentioned previously: the Graphical Form Designer (GFD), Visual Basic for Applications (VBA), or Open Forms.

The Graphical Form Designer (GFD) is used to build the GUI forms that will be presented to the user. Typically, a form comprises a number of fields that hold different values for each case. This set of fields may include *pre-defined fields* such as the current date, current case, and step of the current case, and *custom fields* that are defined for a particular procedure and that are relevant only for that procedure. Fields may be mandatory, optional, read-only, computed by a formula, or even hidden. The GFD also allows forms to be dependent upon run-time conditions, changing the appearance and contents of fields based on rules or data from a specific case.

As a more flexible alternative, Staffware provides a VBA development environment that is similar to Microsoft Visual Basic. This allows building user interfaces that are completely customized according to the tasks at hand, and that exhibit the look and feel of regular applications. The VBA development environment is embedded directly into the Graphical Workflow Definer (GWD) so that run-time solutions can be built as the procedure is being defined. In practice, though, there is a sharp contrast between the simplicity of the GWD and the effort required to build new applications from scratch, while ensuring a consistent handling of all data fields across several procedure steps. Staffware provides powerful debugging capabilities, which further suggests that VBA, being is the most flexible option, is perhaps the most troublesome option too.

The Open Forms is another alternative to GFD forms, but using existing external applications. The Open Forms facility makes no assumptions about what software is used to manipulate field values, requiring only that the external application should be able to call a Dynamic Link Library (DLL). This is necessary for closing the loop and returning the results back to Staffware. The external application is started up from Staffware as soon as the work item is opened from the work queue. The application can be initialized with Staffware data or it can acquire the data by invoking the DLL. After the user interacts with the external application, the DLL is again used to update data in Staffware. Then the external software notifies Staffware when it is closed. Clearly, Open Forms cannot avoid a certain need for integration between Staffware and the external applications.

3.4.1.7 Invoking applications

There are several mechanisms that Staffware can use to invoke external applications. One is the automatic step used in the GWD to represent the invocation of an external program. When defining an automatic step, the full path to the program must be specified as well as the necessary permissions for running the program, and the corresponding executable file must exist. The program is passed two command-line arguments. The first argument is the name of a temporary file which holds the input field values for the step. The second argument is the name of a file the program can create if data needs to be passed back to Staffware. Each line of this output file should contain a field name followed by its value, separated by a comma.

Besides external programs, an automatic step may invoke a Staffware script. A script is a collection of statements, a sequence of operations that the Staffware Workflow Server should perform. Staffware has a simple language for writing scripts, where statements are entered one per line and most of the statements consist of assignment expressions and calls to functions that invoke external or client programs. The Staffware script language has constructs such as conditions and loops, and it uses an extensive set of functions such as conversion, environment, file, date/time, text manipulation, external program, and validation functions. Scripts can be called at other points in the system, and they can be associated with form fields, for example.

Another kind of step that invokes an external application is the Open Client Step (OCS). This type of step allows the use of an alternative client system rather than the Staffware Workqueue Manager. The step passes the work item to the external application by calling a gateway program. The case is then held until the external application releases the work item and notifies this release through that same gateway. There are two external applications that Staffware is able to invoke from an Open Client Step: Microsoft Exchange and Lotus Notes. In both cases, the work item is passed out to the external client through the gateway, rather than being placed in a work queue. The user opens the work item just as any other e-mail message and is presented with a form which is functionally equivalent to a GFD form. The form includes appropriate buttons for returning the results back to Staffware.

Despite these possibilities, the most sophisticated way to invoke external applications is by means of Staffware Brokers. Staffware Brokers are programs that behave like a human user because they have their own work queue from where they consume work items, as suggested in figure 3.17. A broker is permanently running, either on the Staffware Workflow Server or on a separate application server, picking and handling work items from its work queue. When the broker opens a work item, it reads its input information, calls external applications or components, then receives the output data and releases the work item. Brokers are highly versatile programs that are able to invoke CORBA, COM[38], and JavaBeans[39] objects, they can interact with the messaging services, interact with legacy systems through RPC, and even convey data to mobile phones using WAP. The Staffware Broker exhibits several advantages when compared to other options because it exchanges data with the server through a regular work queue, while other application invocation mechanisms must employ intricate ways to return the results back to the server.

[38] http://www.microsoft.com/com/

[39] http://www.java.sun.com/products/javabeans/

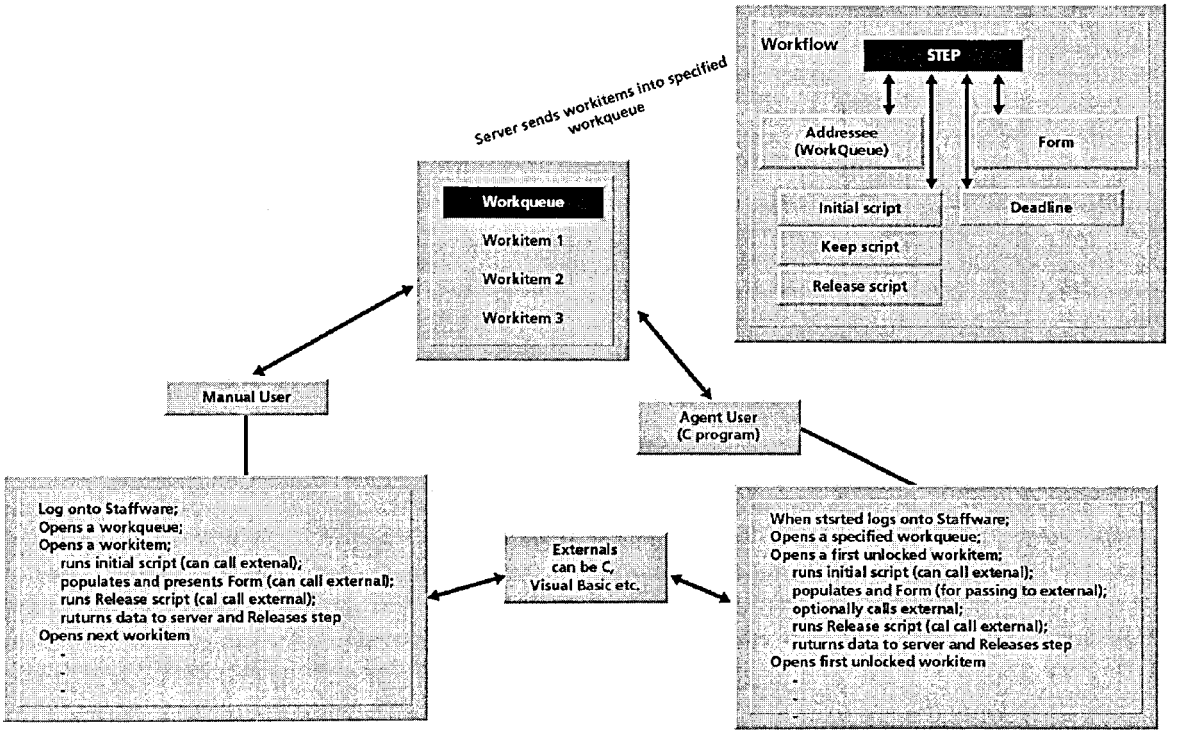


Figure 3.17: Manual users versus Staffware Brokers[37]

3.4.1.8 Workflow client applications

Whereas some applications are to be invoked by the Staffware Workflow Server, a different kind of applications - in particular those which are workflow-enabled - need to invoke capabilities from the Staffware Workflow Server. These are workflow client applications, or enterprise clients as depicted in figure 3.14. In the past, the only way enterprise clients could invoke the Staffware Workflow Server was through the Staffware Application Layer (SAL) and the File Interface Layer (FIL). Recently, Staffware devised a new and more structured way of accessing this same functionality by means of the so-called Staffware Enterprise Objects (SEO). The SEO are an object-oriented API that allows enterprise clients to be built using standard development environments. The SEO API is available in C++, COM and Java, and gives access to the Staffware Workflow Server from virtually any client environment, including both application front-ends (Visual Basic, Oracle Forms, etc.) and browser-based front-ends (JavaScript, ASP, etc.).

The SEO architecture comprises a set of objects that encapsulate all of the Staffware Workflow Server functionality. The primary object types exposed by the SEO are:

- Enterprise - represents all Staffware nodes in the enterprise. It provides access to information about queues and procedures across multiple Staffware nodes in the enterprise. All other objects are exposed as attributes or returned by methods of this object.

[39]Reprinted with permission from Staffware PLC

- Node - represents a specific Staffware node and contains information regarding all users, groups, procedures, and queues that are owned by the node. This object is connected with a specific Staffware Workflow Server.
- User - represents a Staffware user, it contains information about that user, and it can be used to log a user into a Staffware node.
- Procedure - represents a Staffware procedure, containing information about steps, step fields, list of cases, and audit trail. It can be used to start and manipulate the state of cases.
- Case - represents case specific information such as the list of field values. It can be used to notify external events to the Staffware Workflow Server.
- Queue - represents a list of work items that allows applications to present their view on the queue.
- Queue Item - can be used to open and release work items.

3.4.1.9 Audit data

Staffware requires the use of a particular database system for storing all workflow data - including audit trails. The Staffware Executive Information System (EIS) is used to retrieve audit trails from the workflow database and to create management reports from those data. The EIS requires three steps to create these reports. First, the desired cases and fields must be selected from the GWD. The EIS will then perform several queries on the workflow database concerning those cases and fields. The third step is the creation of the report based on query results. Typical reports include performance indicators such as the work load on groups and individual users and how long a given process and its constituent steps take to complete.

3.4.1.10 Compliance with standards

In many respects, the architecture of Staffware resembles the Workflow Reference Model. It has a central Staffware Workflow Server that plays the role of a workflow enactment service. It has a modeling tool (GWD), several options for invoking applications (automatic steps, scripts, Open Client Steps, and Staffware Brokers), an interface to workflow client applications (SEO), and communication mechanisms between Staffware nodes. This similarity is no coincidence and Staffware was one of the first products to become seriously committed to endorsing WfMC standards. Staffware has demonstrated support for WfMC's interface 2 (workflow client applications), interface 3 (invoked applications) and interface 4 (workflow engine interoperability). Additionally, Staffware PLC has performed trial developments with interfaces 1 (WPDL) and 5 (audit information).

3.4.2 OfficeWorks

The OfficeWorks suite from ParaRede is an integrated solution for managing electronically all office documents and communications. Based on a client/server architecture, the OfficeWorks suite comprises a set of services to be shared by all users. Client applications at user desktops connect to the OfficeWorks server through the local network by means of TCP/IP. The server provides all services from the OfficeWorks suite: correspondence log, fax, e-mail, address book, and workflow. The first

three services are the foundation of OfficeWorks and the workflow service - called Floway - relies on those services to interact with human resources.

The OfficeWorks suite has strong infrastructural requirements. It requires Microsoft Exchange for e-mail communication, and Microsoft SQL Server as the database system. In addition, the OfficeWorks server must be installed on the central server of a Microsoft Windows NT domain, which OfficeWorks will use to identify, authenticate and configure users and their privileges. Individual OfficeWorks services can be configured by a set of command-line and graphical applications.

3.4.2.1 The correspondence log service

The correspondence log service allows information about all physical correspondence to be archived, searched and retrieved. This information is contained in log records, which in turn are grouped into log books. While log records are used to maintain information about single correspondence items, different log books are used for each kind of correspondence. For example, there may be a log book for sent correspondence, and another for received correspondence. One log book will contain several log records. In general, a log record contains information about a letter such as its date, the sender name, the sender address, the subject and an optional list of other documents associated with that item. The contents of a log record may be distributed among any number of users, by means of the e-mail service or the workflow service.

3.4.2.2 The fax service

The fax service provides and easy way for sending fax documents from any user desktop. For this purpose OfficeWorks requires an Elefax server, i.e. a separate PC with one or more fax cards that is able to connect to multiple phone lines. The fax cards are supplied by ParaRede as well. The fax service has several parameters that can be configured. First, the number of available phone lines must be specified, as well as which users have access to those lines. Other parameters are the number of attempts that the fax service should use if errors occur during transmission, the priority for each fax, and the front cover that should be used when sending faxes.

The fax service allows users to send faxes with a short message on the front cover, faxes containing only documents, or faxes that contain both message and documents. It is also possible to send faxes to mailing lists. When sending a fax, the user can select the recipient from a set of phone books. Global phone books can be accessed by all fax service users, while private phone books are only accessible to its owner. Only administrators may create or change global phone books. In case the fax is sent to multiple recipients, the user is able to select multiple entries from the phone books and it is possible to obtain a customized front cover for each destination. After the recipients are chosen, the user may attach any number of printable files to the fax. Additionally, OfficeWorks allows users to insert signatures in faxes. These signatures are digitized images of hand-written signatures. Typically, a user will have access to its own signature only, but OfficeWorks allows a user to be configured as being able to include the signature of another user in faxes.

For each phone line, OfficeWorks has two folders: one for faxes that were sent and another for faxes that were received through that line. To read files from any of those folders, the OfficeWorks suite includes an Image Viewer, a client tool that is able to read and display several image file formats. Because a fax comprises mainly pages and text, the tool is slightly different from regular image viewers. And because it is integrated with the OfficeWorks suite, it is able to retrieve, for a given fax being displayed, information from the Elefax server such as when the fax was sent or received, how

many attempts were needed, how long the transmission took, were the fax was sent to, or who was the user sending the fax.

3.4.2.3 The e-mail service

The e-mail service comprises a set of functions that go beyond the regular exchange of e-mail messages. Similar to the fax service, the e-mail service has its own server and clients. However, the server relies on Microsoft Exchange and the clients are built on top of Microsoft Outlook. The main functionality of the OfficeWorks e-mail service is a set of functions added to Microsoft Outlook that allow the user to send regular e-mails as well as *circular* e-mails. All messages use HTML and the server stores them on the Microsoft SQL Server database.

Regular e-mail messages are sent and received using Microsoft Outlook functionality with a slightly different front-end that is provided by OfficeWorks. The key innovation is the possibility of exchanging circular e-mail messages. A circular message is sent to the first recipient from a list of destinations. Then the message will be sent from that first recipient to the second, from the second to the third, and so on, until it reaches the last recipient. Each recipient represents a specific stage of the circular message. On each stage, the corresponding recipient is able to change the e-mail message - by adding text or attaching more documents - before the message is sent to the following recipient. As a second possibility, the recipient may also suspend the circular message.

When creating a circular message, the user may specify a form to be included in the message body. The user can then compose the message by partially or completely filling the form and by specifying multiple recipients for the e-mail message. Up to this point there is no distinction from a regular e-mail message, except that the multiple recipients will be used as the list of recipients, using the order the user specified. The distinction between a regular e-mail and a circular will become more apparent when the recipients receive the message. As shown in figure 3.18, the upper part of the

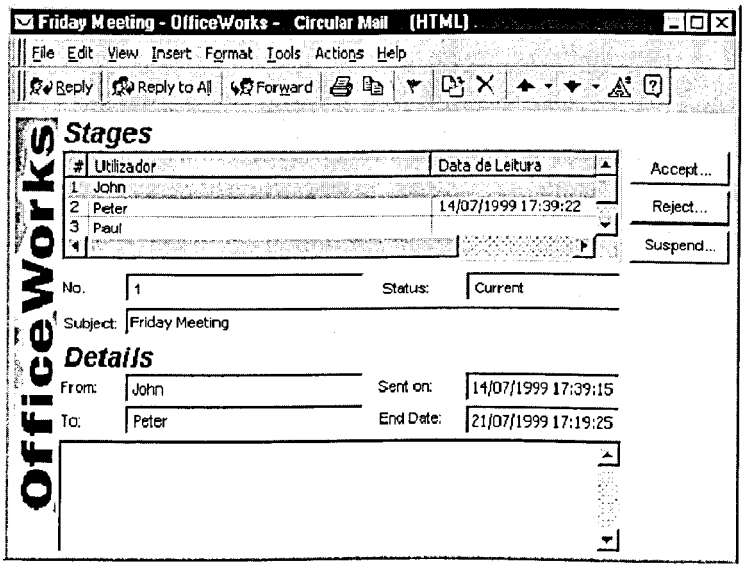


Figure 3.18: Reading an OfficeWorks circular e-mail message[40]

[40]Courtesy of ParaRede

message lists all recipients for the circular e-mail and the dates when those recipients have received and dispatched the message, respectively. For those stages still to be initiated, only the recipient name is displayed. The first row in the list contains the name of the user who created the message, and the date when it was dispatched.

Besides inserting more text and attaching additional documents, it is possible to find out all the information produced in each stage by simply selecting that stage from the list. Other elements give information about the message: the number field is a unique identifier that OfficeWorks generates for each circular e-mail, the status field denotes whether the message is active, suspended or completed, and there is also an optional deadline field. The sender of the message is always the user who completed the previous stage.

When performing a given stage, the user may need to contact one or more users from a previous stage. For that purpose, the user should reply to the circular message. The reply will be sent as a regular e-mail to the specified recipients. This is different from sending the circular message to the following recipient, where the destination field will be automatically specified as the next recipient from the list. If the user suspends the circular message, then the message will not proceed to the following stage. This action may be required if, for example, one document is missing. When suspending a circular message, the user should describe the reason for doing so, and that reason will be saved with the message on the database. The suspended message can be resumed at any time.

When the circular message reaches its last stage, the user is presented with two options: either acknowledge its completion or define new stages for the circular e-mail. As in the previous stages, the user should press the accept button shown in figure 3.18, and then either leave the destination field blank or specify more recipients. If more recipients are specified, these will be added as new stages and the circular message will proceed until the last stage is completed.

3.4.2.4 The address book service

The correspondence log service, the fax service and the e-mail service all deal with a great amount of information concerning contacts, addresses, fax numbers, and e-mail addresses. OfficeWorks provides the address book service in order to manage that kind of information. The address book service is integrated with the previous services in such a way that it is possible to send a fax directly from the address book, query the address book from the e-mail service, or update addresses from the correspondence service, for example.

The address book service has its information structured according to address cards. Each address card contains information such as name, address, phone and fax number. Address cards can be grouped into address lists; this is analogue to the structure of log records and log books from the correspondence log service. Because the number of address lists and address cards may grow indefinitely, OfficeWorks allows searches to be performed across the whole contents of the address book service, or at least across the number of address lists that the user can access. Users of the address book service are organized into groups so that each group has access only to a subset of the available address lists. When creating a new address card, however, all users have access to a common set of tables that contain pre-configured possible values for several address fields such as a lists of countries, districts and ZIP codes. Additional tables of pre-defined address field values may be configured.

3.4.2.5 The workflow service (Floway)

The OfficeWorks services just described provide an infrastructure for communication within and beyond enterprise borders. These services are the enabling platform for a higher-level service - the workflow service, also called Floway. The name comes from OfficeWorks terminology that refers to a process definition as a *flow*, and to process instances simply as *processes*. A flow or a process contains several *stages*, and each stage may contain one or more *tasks*. Each stage from a process is assigned to a single user or to a user group, that are supposed to perform all of its tasks. *Process variables* assume values during run-time and may be used to evaluate *conditions* that influence the execution path of a process. Forms allow users to set the value of process variables, and these forms can be implemented by any application that is able to invoke a set of functions from a DLL that belongs to OfficeWorks.

Floway allows flows to be defined graphically, as shown in figure 3.19, using a small set of constructs:

- Stage - a process activity that can be assigned explicitly to a user group or implicitly to the user who triggered the process. A stage contains an arbitrary number of small tasks, each of which may be performed manually or automatically.
- Condition - represents a boolean expression where process variables are compared with specified values. A condition may evaluate to true or false, so there are two alternative paths arising from this construct.
- Junction - is a synchronizing point for several stages. Process execution will only proceed from this point onwards when all of the previous stages have been completed.
- Knot - allows different paths of execution to be joint or split without having to be synchronized.

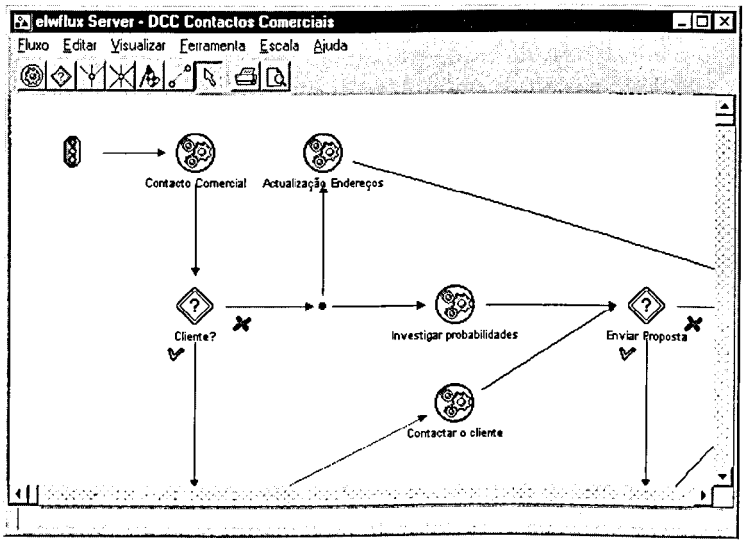


Figure 3.19: Front-end provided by Floway to edit flows[41]

[41] Courtesy of ParaRede

- Terminator - specifies the end of the flow.

These constructs allow several kinds of behavior to be specified, even those behaviors that are prone to inconsistencies. For this reason, the flow editor requires each flow to be tested against errors before creating any processes from that flow. Before that is successfully done, a flow is tagged as being under construction. The flow editor saves all flows in the server database and is able to export them in WPDL format. Therefore, Floway implements WfMC's interface 1.

Whenever appropriate, users are able to access flows and create process instances. For example, if there is a flow for authorizing a visit to a customer's site, then a process will be created from that flow whenever such a visit must take place. The user who creates the process is called the *process originator*.

When defining a flow, each stage on that flow may be assigned explicitly to a particular user group, or implicitly to the process originator. An optional deadline for the stage may be specified. Each stage contains an arbitrary number of small tasks, which in turn may be mandatory or optional. The user can perform these tasks by any order. Tasks have a description that informs the user of its purpose, in case the task is to be performed manually. Additionally, a task may require an automatic operation to be performed.

Floway allows several possibilities for the automatic operation associated with a task. The first possibility is invoking an external command or application. It is possible to specify the command-line parameters to be passed to the application either as fixed string values or as values taken from process variables, and it is possible to specify a process document that the external application should open. The second possibility is launching a form. Because a form can be any application that is able to call functions from a DLL, the command-line parameters also apply, but not the option of associating a process document since the purpose of the form is to allow the user to set the value of process variables. The third possibility is to send an e-mail where the destination, subject and other fields can

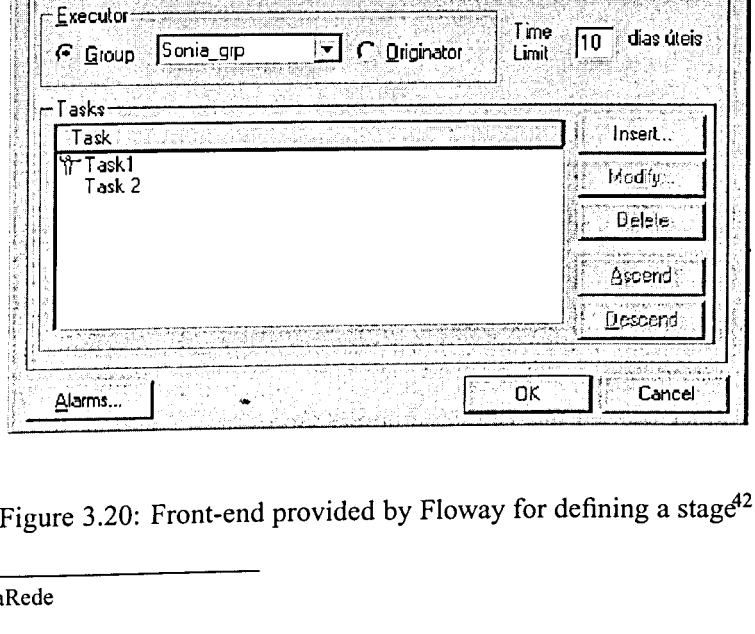


Figure 3.20: Front-end provided by Floway for defining a stage[42]

[42]Courtesy of ParaRede

BIBLIOTECA F.E. PORTO

assume values taken from process variables. A fourth possibility is to send a fax, which simply opens a fax service screen for introducing the details about the fax to be sent.

Process variables are always referred to by their name and they can have one of several types, namely text, date, integer, float, boolean, or even an OfficeWorks signature. Variables can have a default value according to their type. In case the variable is a signature, whenever the variable shows up in forms the user must enter the password in order to assign a signature to the variable, or retrieve the signature from the variable. User authentication is based on the same user privileges for accessing fax service signatures. Besides this authentication mechanism, each user may be configured to have no access, read access, or write access to any process variable.

In a similar way to other OfficeWorks services, user privileges are enforced throughout the whole functionality of Floway. In order to create or edit flows, to initiate processes, to modify or delete processes, and to perform stages and its constituent tasks, a user must have the appropriate privileges. User privileges may be configured globally and applied simultaneously to all functions within Floway, or they can be configured for each flow, for each process, or for each process variable. Users can be organized into groups that share the same privileges.

When a stage is assigned to a user group, one member from that user group is expected to accept it. When that happens, the stage is added to the list of stages for that user, as shown in figure 3.21. For each stage the list of tasks is displayed in the lower part of the window, as well as the documents associated with each task. The user can open each task so as to read its description and to find out if any automatic operation is associated with it. In this case, the user can run that operation. After all tasks are executed, the stage is complete. If the stage becomes delayed beyond its deadline, Floway is able to generate an alarm that is be sent to the user, to the user in charge of the group, or to the process originator. The alarm is sent by e-mail or by popping up a window at the recipient's desktop.

The user may also delegate the stage to another user from the same group. The stage can be delegated with approval, meaning that the stage must be approved by the first user who delegated the

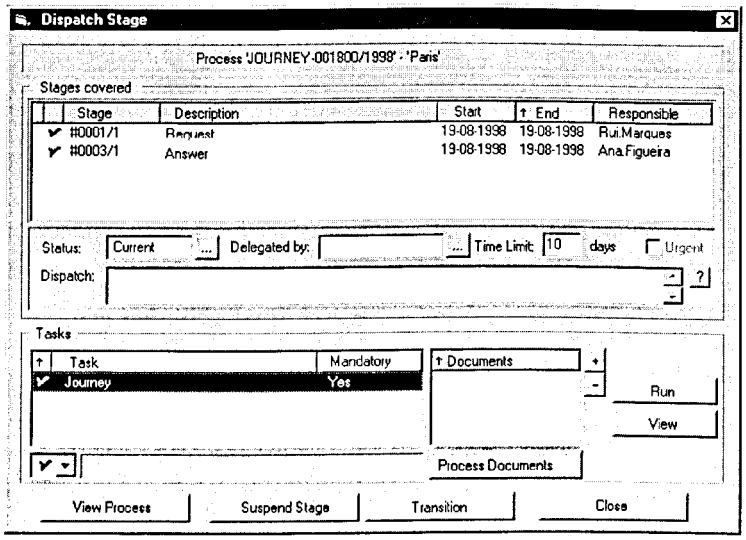


Figure 3.21: Front-end provided by Floway for users to manage stages[43]

[43]Courtesy of ParaRede

task, before the process can proceed. A stage can be delegated more than once, and Floway keeps track of all delegations.

3.5 Research applications

Commercial workflow products are increasingly adopting the latest Internet-related technologies such as browser-based client applications and task delivery via e-mail. In fact, figure 3.12 on page 86 shows that most commercial workflow systems available today make use of HTTP and e-mail protocols. Despite this fact, most (if not all) workflow products are clearly focused on the integration and coordination of internal business processes and, in general, provide no support for the integration of business processes that go beyond enterprise borders. This challenge has received more attention from academia and research communities, which in recent years have been advocating the need and benefits of new organizational paradigms such as the virtual enterprise.

As a matter of fact, workflow management is often an important ingredient in research projects that address the infrastructural requirements of virtual enterprises, where integrating and coordinating inter-enterprise business processes is a major issue [Grefen et al., 2000]. In this case, workflow systems are typically associated with distributed systems or Web-based systems so as to provide distributed process management capabilities. This is the most important trend concerning this work.

Another recent trend in workflow research projects is the development of new workflow management systems strongly based on Internet-related technologies [Petrie and Sarin, 2000]. This trend - called "Internet-mediated workflow" - focuses on demonstrating that the Web can be used as an infrastructure for workflow management. Research prototypes that have resulted from this trend show that the Web can significantly improve the capabilities of current workflow management systems.

3.5.1 System-centered workflow

The first research efforts on workflow management, however, have been devoted mainly to the transactional capabilities of workflow management systems [Sheth and Rusinkiewicz, 1993]. To a great extent, this trend is an influence from database management, which was already a very active and promising research field when workflow management emerged. Later, this focus shifted towards object-oriented design of workflow management systems [Miller et al., 1996], while transactional capabilities were discussed in combination with distributed workflow architectures. These system-centered issues belong to classic research on workflow management.

3.5.1.1 TriGSflow

An emblematic reference concerning classic research on workflow management is TriGSflow [Kappel et al., 1995]. TriGSflow is an object-oriented workflow system built on top of TriGS which, in turn, is an extension of an object-oriented database. TriGS implements the Event-Condition-Action (ECA) rule paradigm and TriGSflow uses this paradigm to execute activities that are regarded as single transactions. Basically, the ECA paradigm states that if a given *event* occurs and a certain *condition* is true then a pre-defined *action* should be executed. When executing a particular process definition, the event is usually interpreted as the completion of the previous activity and the condition may be used to define branching decisions.

In reality, TriGSflow applies ECA rules in two further areas: resource selection and task list management. TriGSflow distinguishes between human and application resources and allows three

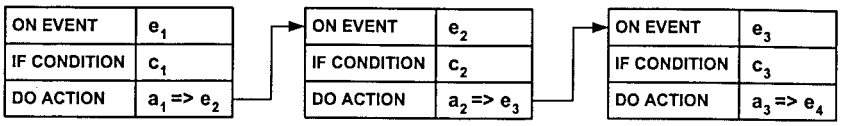


Figure 3.22: Activity sequence according to ECA rules

resource assignment methods: implicit allocation if the activity is assigned to an internal application, explicit allocation if the resource is assigned when the process is defined, and run-time selection if the resource is selected by an ECA rule that is executed before the actual activity is performed. The task list for each resource is also managed by an ECA rule: whenever a resource completes a task, TriGSflow raises an event that will lead to the evaluation of a condition. That condition is whether or not there are more tasks in the task list waiting to be performed. If so, the corresponding application resource is invoked or, in the case of a human resource, TriGSflow will print a message on the user's desktop and wait until the user signals that the task has been completed.

3.5.1.2 SWORDIES

Another research project that makes extensive use of ECA rules is SWORDIES [Endl et al., 1998]. However, SWORDIES extends ECA rules into ECAA rules (Event - Condition - Action / Alternative Action). The meaning of this rule is equivalent to the ECA rule except for the fact that if the condition evaluates to false then an alternative action may be performed. The purpose of this construct is to allow branching within one single rule.

The focus of SWORDIES is quite different from that of TriGSflow. SWORDIES realizes that there is a life cycle for business process models and that this life cycle comprises four phases: the *requirements analysis phase* when processes are examined and documented, the *system analysis phase* when processes are expressed as formal specifications, the *architecture definition phase* when processes models are integrated with a logical component-based architecture of the workflow management system, and the *implementation phase* when the process models are brought into execution. The goal of SWORDIES is to provide a uniform process modeling environment across these four phases, and that is the purpose of using ECAA rules.

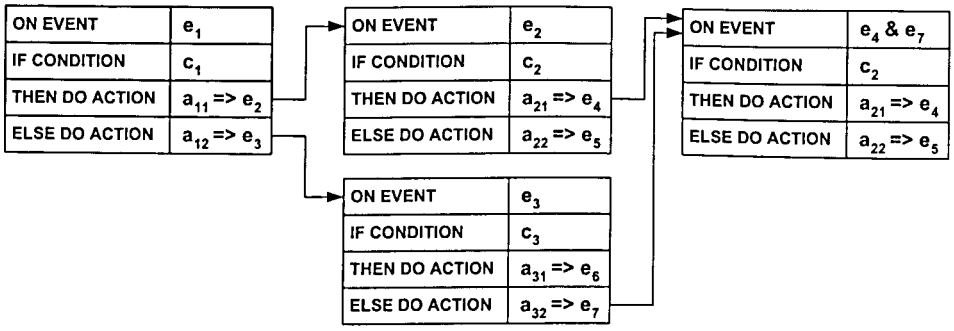


Figure 3.23: Activity sequence according to ECAA rules

In order to achieve this uniform modeling environment, SWORDIES allows two other extensions to the ECA rule. The first extension is to include *actors* (i.e., resources) that are responsible for evaluating the condition and performing the action of the ECAA rule. The same actor may be assigned to both steps, or there may be a different actor for each step. The second extension is to include input and output data in an ECAA rule. SWORDIES represents workflow relevant data according to an entity-relationship model, and allows entity-relationship types to be used as input data for evaluating the condition of the ECAA rule, and as input and output data for each action of the ECAA rule. These constructs can be used in order to refine process models throughout the model life cycle. For that purpose, SWORDIES adopts a refinement mechanism that allows ECAA rules to be refined hierarchically, i.e., one ECAA rule that is used at a higher-level of abstraction (e.g. order entry activity) may be refined into a sequence of lower-level ECAA rules (e.g. operations on the information system).

3.5.1.3 EvE

Just like TriGSflow requires TriGS that implements ECA rules, SWORDIES relies on a run-time execution system called EvE [Geppert et al., 1998]. EvE is an event-driven and a distributed workflow engine. EvE has a multi-server architecture where servers connect to a central run-time repository where EvE stores all event types and ECA rules. All workflow clients interface their corresponding server by means of an EvE adapter that conveys events to the local server. When a server receives an event, it forwards the event to the other servers so that all servers can execute the ECA rules that are triggered by that event. A typical condition in an ECA rule will determine which resource should be invoked, and the action typically notifies that resource by updating the corresponding task list or by invoking an external application program.

Workflow execution, according to EvE, has four distinct phases. In the first phase - event generation - an external application or a workflow client generates an event, which can be interpreted as the triggering event for a certain process. In the second phase - event detection and logging - the event is matched against the configured ECA rules in order to determine the set of rules to be executed. The third phase consists in the execution of that set of rules: evaluating their conditions and initiating their actions. This results in invoking the assigned resources, which will then perform their tasks during the fourth phase - service execution. During this phase, resources generate new events, and EvE resumes workflow execution back from the first phase, event generation. Because these new events will trigger new ECA rules, the output events from some ECA rules are input events to others, and a process is effectively executed as a sequence of ECA rules.

3.5.1.4 APRICOTS

Although TriGS and EvE implement ECA rules based on database transactions, other authors have realized that workflow activities, due to their possibly long execution time, pose further requirements on transactions beyond the classic ACID properties (atomicity, consistency, isolation, and durability) [Alonso et al., 1996]. This issue has been the focus of the APRICOTS project [Schwenkreis, 1993]. APRICOTS is a prototype implementation of a workflow management system based on the ConTract model [Wächter and Reuter, 1992], which is basically a system that supports long-lived transactions that are composed of smaller, ACID-compliant transactions. These elementary and arbitrary ACID transactions are called steps, and a sequence of steps makes up a control flow description called script.

The APRICOTS system comprises six different modules. The ConTract-Manager runs the scripts and provides the transactional environment for their execution. This module requests all the individual transactions of a script to be performed by another module, the Transaction-Manager, but it is the ConTract-Manager which is responsible for transaction recovery of a script. The Step-Server is a user-programmed module that contains the executable code for all steps. This code, which defines individual transactions, may include interaction with other modules of APRICOTS. Whenever the Step-Server needs to interact with external systems, it does so through the Resource-Manager module which interfaces databases and external applications. The User-Interface module allows the user to visualize scripts in a graphical form and to monitor the activity of all other modules. The Communication-System module is an RPC-based system that enables communication between all modules.

Although the user can edit scripts based on simple graphical constructs, APRICOTS uses an internal representation format which is called predicate-transition-net (PTN). The PTN is somewhat inspired in Petri nets [David and Alla, 1992] since it has places (called *spots*), transitions and tokens; the difference is that the connection from spots to transitions is attained by predicates, which are equivalent to boolean conditions. A step from a script is represented on a PTN by a spot. APRICOTS has therefore two levels of representation for a process definition: a high-level, simple representation of scripts that the user can easily read and maintain, and a low-level, PTN representation that is filled with additional proprietary information that is needed for executing the transactional scripts. The concept of having two representation levels is used in several research projects.

3.5.1.5 ADEPT

Besides addressing transactional capabilities, the ADEPT project [Dadam and Reichert, 1998] considers improvements in several other aspects of workflow management systems. The experience of the ADEPT team in applying workflow management systems in a clinical environment has led to the identification of possible improvements in several areas, namely process modeling, supporting process changes at run-time, enforcing temporal relationships between activities, interference factors between activities, easier user interfaces, and support for multiple workflow servers. From these areas, ADEPT has devoted most of its effort to enabling dynamic workflow changes [Reichert et al., 1998]. ADEPT explicitly dismisses the use of Petri nets and ECA rules and prefers instead a proprietary, block-structured description language, which leads to a particular definition of process consistency. The ADEPT workflow management system prototype is able to enforce this consistency when inserting or deleting ad-hoc activities at run-time.

The ADEPT project addresses two kinds of run-time process changes. The first are changes to the process definition. In this case, changes will only affect new instances created from the process definition, while currently running instances remain unchanged. This means that it is possible to run, at the same time, process instances that follow the new process definition and process instances that follow the old one. The workflow management system must therefore maintain different versions of the same process definition. The second kind of run-time changes are changes in the process instance itself. These may be required to handle exceptional circumstances, or to model activities that are only known at run-time. In this case, not every change is allowed since activities that have already been completed cannot be removed, and new activities cannot be inserted in portions of the process that have already been performed. In other words, only the activities that are yet to be performed can be changed. Then the same consistency checks that are applied at build-time must be applied at run-time too.

An additional characteristic of the ADEPT workflow system is that process execution can be distributed among several workflow server (or engines). Each server is responsible for the execution of a partition from the original process instance. This is similar to interoperability mode 3 shown earlier in figure 3.5 on page 70, but in ADEPT a workflow server is usually responsible for contiguous blocks of activities. For example, if a process definition has two parallel branches, the execution of these branches can be assigned to different workflow servers. In general, ADEPT requires servers to synchronize their partitions only when they become completed, and not while they are being executed. The partitions are usually chosen so that communication between the different workflow servers is minimized.

Due to the possibility of distributed execution, handling run-time changes becomes even more difficult. If the changes are confined to a single partition then the assigned workflow server can handle them alone. But if changes affect other partitions too, the workflow server must first retrieve the execution status from the corresponding servers, and only then can it check the consistency of the desired change. Afterwards, if the change is actually applied to the local partition, the local server must notify the other servers so that they can change their partitions too. When dealing with run-time changes, the ADEPT prototype assumes that the execution of the process instance is controlled by a single workflow server, which is equivalent to having a single partition. Run-time changes to multiple partitions require further research [Reichert et al., 1999].

3.5.1.6 MENTOR

The MENTOR project [Wodtke, 1997] uses state charts in order to model business processes and has also developed its own techniques for ensuring the consistency of process definitions. However, the main issue about MENTOR is that it was the first workflow system of its kind: it was built according to a distributed, client/server architecture and it relies on a transaction processing monitor (or TP monitor) [Edwards, 1999] to communicate between workflow servers. The use of CORBA in MENTOR was due to the need to cope with the heterogeneity of information systems available in enterprise environments. Additionally, MENTOR was intended to be scalable and fault-tolerant architecture, which led to the use of a TP monitor to manage the use of computational resources and to provide transactional support for workflow activities.

As shown in figure 3.24, the MENTOR architecture is able to include several workflow servers, each of which contains a workflow engine that is able to interpret state charts. During workflow execution, the workflow engine is able to invoke any application for which an activity stub - which is, in essence, an IDL (Interface Definition Language) wrapper - has been provided. In other words, the workflow engine is able to invoke any application that implements a pre-defined set of CORBA interfaces. Each workflow server also contains a Communications Manager, which is an interface between the workflow engine and the TP monitor, allowing workflow engines to distribute the execution of process instances among themselves. This distributed execution is attained by specific techniques of dividing state charts into a set of partitions [Wodtke and Weikum, 1997].

Each workflow engine executes a certain process partition and invokes the applications corresponding to each process activity, using the activity stubs. The Object Request Broker (ORB) maps the calling parameters given by the workflow engine to the actual calling parameters for the invoked application. The Log Manager keeps track of every action taken by the workflow engine so as to be able to recover its state on system failure. Whenever an activity is ready for execution it is sent as a work item to the Worklist Manager. Based on the required role, the Worklist Manager will assign the activity to the appropriate participant, taking into account information such as role resolution policies

and scheduled vacations. Similar to the Log Manager, the History Manager is a bookkeeper for all workflow executions but its purpose is to store information for immediate and long-term monitoring.

Whenever the workflow engine begins or finishes the execution of a workflow activity (workflow server 1 in figure 3.24), an update message is propagated to the Communications Managers so that other workflow servers are notified of this change in state, since the process definition may be partitioned across several workflow engines. The TP monitor determines the list of workflow servers to be notified and sends all update notifications within a transaction so that the states across every workflow server and in the Log Manager are maintained consistent. The actual communication between workflow servers is attained by means of message queuing mechanisms provided by the TP monitor. At each recipient workflow server, the TP monitor creates a new transaction; within this transaction it reads the update message from a local message queue and forwards the message to the local Communications Manager (workflow server 2 in figure 3.24). The Communications Manager passes these message to the target workflow engine and, based on the received data, local state information is updated.

The MENTOR architecture distinguishes between those components that are associated with the workflow engine (Communications Manager and Activity Stubs) from those that are external to the workflow server (Log Manager, Worklist Manager and History Manager). A third kind of application components are the heterogeneous application components (invoked applications) that are available in the enterprise environments. Realizing the challenge of integrating all these components into a common workflow-based approach, the developers of MENTOR came up with a lighter version of that system, called MENTOR-lite [Muth et al., 1998]. MENTOR-lite aims at integrating all those application components in a stepwise manner, while distinguishing between kernel functionality that is intrinsic to the workflow engine and functionality, such as history management, that should be built on top of that kernel.

Indeed, the need for this kind of workflow engine kernel is one of the key findings of this work, as will be discussed later on. MENTOR-lite implements this kernel functionality by isolating the workflow engine functionality from the former MENTOR system. Therefore, MENTOR-lite is also

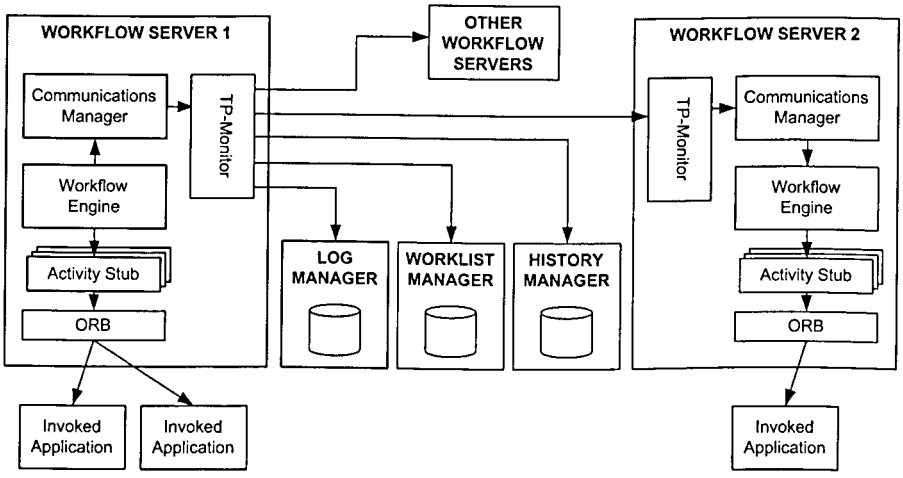


Figure 3.24: Architecture for the MENTOR project[44]

[44][Wodtke, 1997]

based on state charts. The result is that this workflow engine functionality can be reused in other system components, particularly in worklist management [Weissenfels et al., 1998]. When assigning participants to activities, assignment strategies can be described by process definitions, so as to enable flexible assignments based on virtually any criteria, such as role-based assignments, assignments based resource load or assignments based on workflow relevant data.

Besides this enhancement, MENTOR-lite enables extensions to the kernel to be easily implemented based on a single interface to the kernel [Weissenfels et al., 1998]. Through this interface, the kernel's functionality is available to all extensions, allowing for example decentralized execution of assignment strategies or decentralized history management with no additional implementation effort. The same holds for extensions enabling interoperability with other workflow management systems. Recently, MENTOR-lite has been improved to include an infrastructure based on XML and Internet technologies [Shegalov et al., 2001] rather than the original MENTOR infrastructure based on CORBA. This approach reflects the recent trend towards Internet-mediated workflow.

3.5.1.7 WIDE

The WIDE project [Grefen et al., 1998] has also devised a distributed, CORBA-based workflow system architecture. However, the WIDE architecture comprises three layers. The lowest layer refers to a commercial database system (Oracle). The middle layer contains the workflow engine supported by specific technology developed during the project, namely extended transaction management and active rule support. The interface between the database system and the workflow layer is achieved by an interface - called Basic Access Layer (BAL) - that maps object-oriented operations from the workflow layer onto relational operations on the database system. The upper layer consists in a client application that provides a user interface to the workflow engine. The three layers are depicted in figure 3.25.

The WIDE project concerns mainly the development of extended database support for workflow management. WIDE has focused particularly in developing certain database technologies, namely

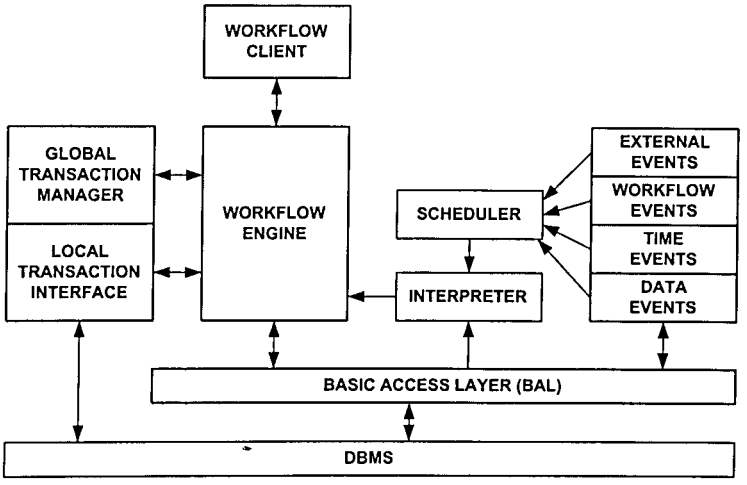


Figure 3.25: Architecture for the WIDE project[45]

[45]adapted from [Grefen et al., 1998]

extended transaction management and active rule support. Regarding transaction management, it is further divided into two layers. The Global Transaction Manager is responsible for certain sequences of activities which, in case of failure of one of them, must be rolled back to a consistent process execution point. These sequences of activities are called business transactions and their execution requires all constituent activities to be performed successfully. The Local Transaction Interface is responsible for implementing the low-level, database transactions that pertain to each particular activity.

Regarding active rule support, the WIDE project aims at providing reactive behavior for workflow management systems in order to cope with four different types of events: external events, workflow events, time events and data events. WIDE uses a method similar to ECA rules to specify the execution of active rules, which are stored in the database system. Upon occurrence of one of those events, the Local Transaction Interface creates a transaction to store these events in the database system. Then, from time to time, a Scheduler is invoked that retrieves the events from the event database and matches them against the available rules. When a match is found, an Interpreter is invoked to evaluate the rule condition, which will determine whether the rule's action is to be executed or not.

An additional result from the WIDE project was the development of a workflow modeling methodology. This methodology requires three different models to be developed - the organization model, the information model, and the process model:

- The purpose of the organization model is to describe the organization entities that are associated to business processes, their position within the organization structure, and their assignment to individual process activities. The organization model enumerates the roles that are required for each process activity and how these roles are assigned to specific entities. Some entities may be able to substitute others, and there are supervising entities as well. There are two possibilities for the assignment of tasks to these entities: either the task is directly sent to a specific entity, or the task is sent to a common worklist from where an entity will pick the task to its own worklist. Some assignment criteria may also apply, such as choosing the entity with less workload or choosing the entity which is the supervising entity.

- The information model describes three types of information items: variables, document information, and temporal information. Variables are essentially workflow relevant data, data that store information relevant to the model and can be used to control the flow of execution. An innovative feature of WIDE is that these variables can be shared across process instances. Their value can be retrieved by any process instance, but it can only be set by external applications. Document information can be of three types: form, document or folder. A form is a set of data fields whose values can be controlled by the workflow management system, and which is to be presented to the user during task execution. A folder is a combination of any number of documents and forms.

  Temporal information is included in the information model in order to describe time events. These time events signal that a particular instant has been reached, that a time interval has elapsed, or that a periodic time event has been issued. Temporal information can be used in process definitions to raise execution exceptions, to issue timeouts and to introduce wait periods during process execution.

- The third model required by the WIDE methodology is the process model, which uses modeling elements such as tasks and connectors. When the workflow engine creates a new process instance, the Interpreter will determine the initial tasks to execute. Besides other attributes, a task contains a reference to a required role that will be used to assign that task to a specific

entity, according to the organizational model. The task will be sent to the user together with the information elements described in the information model. The user has different options for completing the task. One of those options is, as in OfficeWorks, to delegate the task to another user.

The WIDE process model includes several constructs to connect tasks to each other. These constructs include conditional and mutual exclusion split connectors, total and partial join connectors, cyclic connectors, and wait connectors as specified by the temporal relationships expressed in the information model. While most tasks are atomic, WIDE defines a multitask as a task that comprises several individual tasks that perform the same job in parallel, with some parameter that may distinguish them such as the particular assigned entity. The multitask is useful in cases such as review processes when the same information must be submitted to several users for evaluation. During the modeling phase, the multitask is seen as a single task, and only during the execution phase it is split into different task instances.

There are two additional structuring constructs within the WIDE process model: the *sub-process* and the *super-task*. The sub-process enables modular construction of process definitions, and the same sub-process may be used in different processes. The super-task allows groups of tasks to be executed under the same transaction. The super-task shares some concepts with the sub-process but, contrary to the sub-process, the super-task cannot be reused, since it is a group of tasks specific to a certain process. These two constructs are treated differently also in distributed process execution. In fact, and similarly to the MENTOR project, WIDE allows a process to be executed on several workflow engines, although with some limitations when compared to MENTOR. For example, transactional control in WIDE is confined to the local workflow engine, while MENTOR is able to manage distributed transactions. According to WIDE, a sub-process can be executed remotely and separately from the parent process, while a super-task must be executed in the same local engine.

3.5.1.8 WASA

Another project that has developed a CORBA-based, distributed workflow system architecture is the WASA project [Medeiros et al., 1995]. The WASA project aims at providing workflow support for a variety of domains, particularly the domain of scientific applications. The analysis of requirements for scientific applications such as molecular biology and geo-processing led to the concept of *scientific workflows*: processes that are clearly oriented towards reproducing scientific results. WASA focuses on applying flexible and platform-independent workflow management to these processes. By "flexible" it is meant that it should be possible to dynamically change processes at run-time. Platform-independence is the other goal of WASA: the development of a workflow system that can run on virtually every platform.

The architecture for the WASA project, as illustrated in figure 3.26, has similar features to that of WIDE: the workflow management systems and its associated tools rely on a layer of extended database functionality. The first workflow management system prototype developed within the WASA project stored process definitions in the relational database and relied extensively on Java and JDBC technology to achieve platform-independence. The client applications were Java applications or Java applets that could be displayed in a Web browser. This prototype allowed changes to be inserted in running processes, as long as these changes were inserted only in the group of tasks that were yet to be executed at a given point in time. These dynamic changes are possible because the workflow engine does not retrieve the whole definition from the database when the process instance is created.

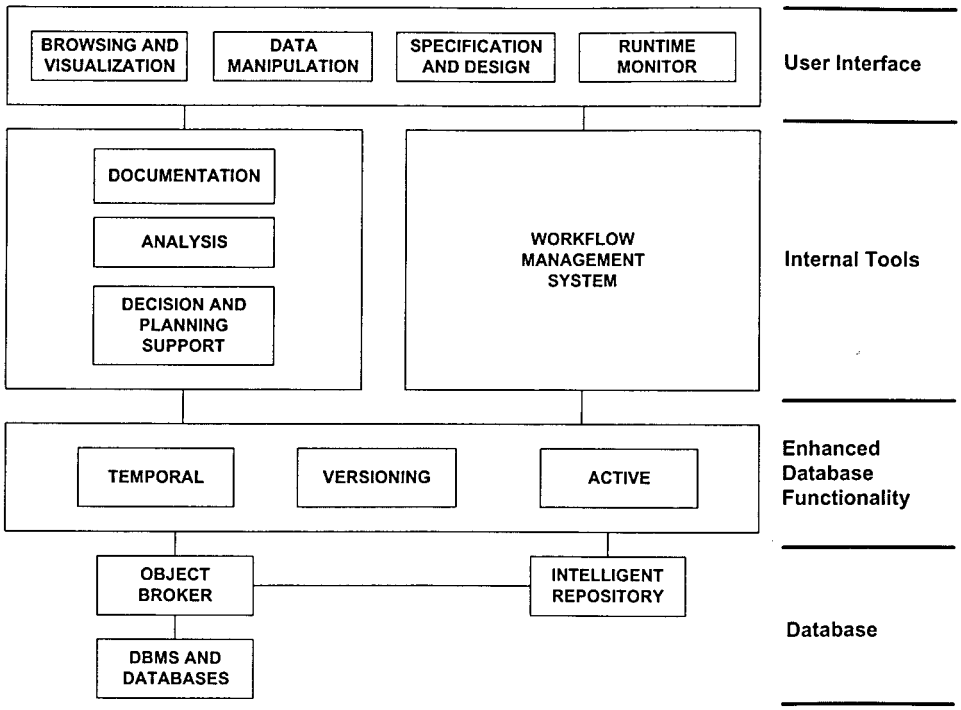


Figure 3.26: Architecture for the WASA project[46]

Instead, the workflow engine loads each process activity from the database only when all previous activities have been completed.

After this first prototype, a second WASA prototype was built, this time relying on a CORBA-based infrastructure. This second prototype has improved all functionality from the previous prototype, including process modeling capabilities. Processes are now modeled using nested graphs, where nodes represent activities or sub-processes and edges represent control and data relationships between those constructs. In a nested graph, the data inputs for a parent node represent data inputs of the child nodes, and the same applies to data outputs, as shown in figure 3.27. At a given point during run-time, changes can only be applied to those nodes whose execution has not yet begun, and which have no control connections to nodes that have been already completed. This is also the rule for inserting new nodes at run-time, an especially useful feature in scientific workflows, where the complete process is not known beforehand.

The CORBA infrastructure of the second WASA prototype allows the replication of server objects, so that several instances of the workflow engine may be running at the same time. Because these engines can exchange messages with each other, it is possible to execute each graph node at a different workflow engine instance, where the completion of a node results in a start message being sent from one workflow engine instance to following one. WASA describes this as distributed execution, even though it is not a distributed execution among different workflow engines, but among several instances of the same workflow engine. The result is improved scalability of the workflow management system.

[46][Medeiros et al., 1995]

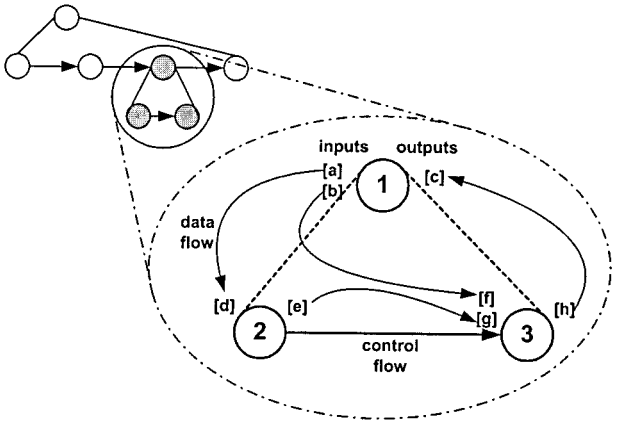


Figure 3.27: Nested graphs for process modeling in WASA[47]

3.5.1.9 METEOR

The METEOR project [Das et al., 1997] has taken a different approach towards distributed execution, as illustrated in figure 3.28. Starting from an architecture where process execution and task delivery are centralized, METEOR has devised an architecture where worklists (or task managers) have decentralized execution capabilities. Instead of a centralized workflow engine, the process definition is passed from task manager to task manager, where each activity instance will be created and performed. The creation of activity instances is attained by scheduler objects which are associated with task managers. Each scheduler controls the life cycle of activities assigned to that worklist. Once a local activity is complete, the scheduler will activate other schedulers so that the following activities are executed, as specified in the process definition. This architecture is supported by a CORBA-based infrastructure.

As in APRICOTS, METEOR has two distinct levels for modeling business processes. One is provided by the Process Modeler, a high-level modeling tool focusing mainly on organizational issues without delving into the procedural details required for run-time. This tool was developed to assist management in describing the processes to be automated. The second modeling level is provided by the Workflow Builder, the necessary tool for creating process definitions that can be used for execution. These process definitions are expressed according to a Workflow Intermediate Language (WIL) that is similar in structure and semantics to the WfMC's WPDL language. The WIL language specifies the activity sequence, the data objects that are transferred between activities, and how each activity is to be performed.

Two additional tools support the development of WIL-compliant process definitions. One is the Data Designer, a modeling tool based on OMT [Rumbaugh et al., 1990] that uses classes and relationships to represent data objects and data operations. Process definitions and the definition of process activities will refer to these objects and operations. The second tool is the Task Designer which is used to specify, for each task, a command line to an external program or an HTML form that is used as the user interface for that task. In fact, although METEOR has a CORBA-based architecture, it relies on the Web browser for rendering user interfaces. At the Web server, a CGI script that is also a CORBA client communicates data and events to the task manager.

[47] [Weske and Vossen, 1998]

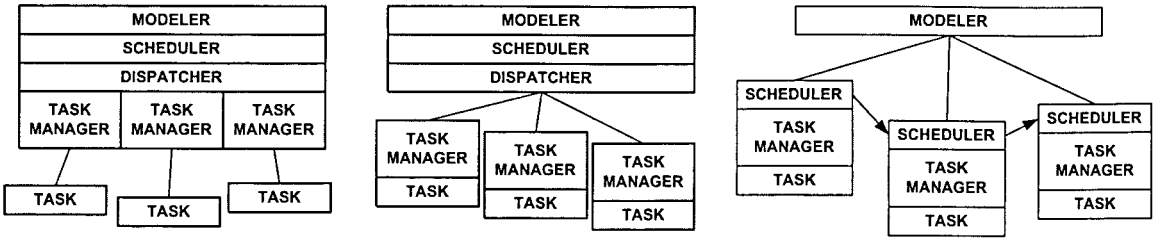


Figure 3.28: Evolution towards the METEOR architecture[48]

Figure 3.28 shows that task managers and schedulers are components that enable the distributed execution of process instances. For each activity in the process definition there is a task manager, a scheduling component, application invocation code, data object access routines, and code for activating other schedulers. METEOR has been devised with a code generator that takes the WIL process definition as input and automatically produces the code for all these components and functionality. Thus, the task manager and its scheduler are not generic components, instead they are highly customized for the execution of a single task. When compared to other architectures, METEOR hardly tolerates any dynamic process changes since this would require the code for task managers to be generated again.

Each task manager is a CORBA object from one of three pre-defined classes that represent transactional tasks, non-transactional tasks, and user tasks, respectively. Although only transactional tasks can be rolled back or recovered in an automatic way, METEOR provides two extra components that support the recovery of other types of tasks. The first extra component is the Local Persistence Store (LPS) that is used for storing the current state of the task manager and of its scheduler. Task managers also log inputs received from other task managers and outputs returned by their tasks. The second extra component is the Local Recovery Manager (LRM). Basically, whenever there is an error or failure in the task manager, the LRM will try to restore the previous state stored in the LPS. If the task was still inactive or if it was already complete by the time the error occurred, METEOR is able to automatically bring the task manager to a consistent state. However, if the task was already under execution its recovery may require task-specific recovery functionality that should be included during the code generation phase.

These features have been implemented in a second METEOR prototype called WebWork [Miller et al., 1998], as opposed to the first prototype which was called ORBWork. WebWork is based on the same architecture but was implemented with Web technologies only. In this second prototype, most functionality is encapsulated in CGI scripts rather than CORBA components. Task managers, which are CGI scripts themselves, communicate through HTTP by exchanging HTML data. When a task is completed, the corresponding task manager produces an HTML page that contains all output data. This output page is then sent to the following task manager. In all other respects, WebWork works just like the ORBWork prototype. The development of this second prototype suggests that METEOR can easily deliver the same functionality on top of different technologies. This is due to the code generating step which, based on the same WIL process definition, can be changed to produce code for a different run-time infrastructure.

[48] adapted from [Miller et al., 1996]

3.5.1.10 EXOTICA

The move from ORBWork to WebWork in METEOR reflects once again the trend towards Internet-mediated workflow. Other workflow projects, however, have employed run-time infrastructures other than CORBA and Web technologies. One such project is EXOTICA [Mohan et al., 1995] which relies on message queuing in order to implement distributed execution. The EXOTICA project was funded by IBM and its purpose was to bring together two IBM products: FlowMark, a workflow management system, and MQSeries, a message queuing platform. Additionally, EXOTICA aimed at introducing improvements in fault-tolerance, scalability, advanced transaction models, and disconnected operation.

FlowMark has a centralized architecture since it relies on a single database to store all workflow-related build-time and run-time information. During build-time, FlowMark uses directed graphs to model business processes, where nodes are activities and arcs represent control and data connections. Activities can be refined into more detailed subgraphs, allowing nesting and composition in process design. In most respects, these modeling features are similar to those of the WASA project, although FlowMark uses its own modeling language called FlowMark Description Language (FDL). A module called Buildtime Client assists the user in creating process definitions according to FDL.

Two additional FlowMark modules implement the run-time functionality. The FlowMark Server module interprets FDL process definitions and controls the execution of process instances. The Run-time Client module provides worklist functionality and the user interface for performing individual tasks. FlowMark supports forward recovery, i.e., FlowMark is able to resume process execution from the point where an error or failure has occurred. The EXOTICA project aims at providing backward recovery support: the ability to implement compensation actions that implement a logical undo operation. For this purpose, EXOTICA has developed the concept of flexible transactions where both the intended and the compensating actions are specified [Alonso et al., 1996].

MQSeries supports asynchronous, message-based communication between applications. This messaging platform implements persistent message queues, which bring several advantages when compared to other infrastructures. This kind of platform allows communication between distributed application components without requiring a connection to be established between those components, and it shields those components from communication failure and recovery. Furthermore, MQSeries protects message queue operations with transactional control so that each message is guaranteed to be handled correctly. EXOTICA uses MQSeries as an alternative to the centralized FlowMark architecture, and as the means to implement distributed workflow execution. Some extensions to MQSeries had to be devised in order to implement special features, such as the ability to migrate a message queue from one node to another. Eventually, the result of EXOTICA was a new product called MQSeries Workflow.

3.5.1.11 MOBILE

In general, all system-centered research projects have focused on the infrastructure and run-time functionality of workflow management systems, and have devoted some attention to the modeling capabilities of those systems. In the case of MOBILE [Jablonski, 1994], the focus has been put mainly on modeling capabilities. The MOBILE project proposes a modeling approach based on four different perspectives: functional, behavioral, organizational and informational.

The functional perspective identifies and describes each step or activity, determines the required application programs and whether or not these rely on legacy systems. The behavioral model describes the control flow and the temporal relationships between steps. The organizational perspective

addresses the existing enterprise resources (humans, machines and applications) and the relationships between these organization objects (hierarchical relationships, role relationships and membership relationships). This organizational model also refers to the methods for resource assignment (e.g. role-based) and to the methods for delivering tasks to resources (e.g. by e-mail). The informational perspective concerns the flow of data between activities, and the translations that may have to take place before data is passed from one activity to the other.

An interesting point about the MOBILE organizational model is that it includes conditions for assigning resources or notifying them, such as, for example, "if price is within the budget assign task to operator, otherwise assign to supervisor". In many workflow management systems, this type of conditions are commonly specified as branching conditions in the process definition (if condition C1 is true do activity A1, else do activity A2). MOBILE, however, treats these conditions as organizational policies that are not particular to a process; rather, they are applied in any process whenever such circumstances apply.

All this information is stored in the MOBILE repository, which can be regarded as a comprehensive class library that spans all objects and relationships from the four models. These objects can be picked from the repository in order to build a process definition in a drag-and-drop manner. At run-time, the MOBILE Execution Engine interprets these process definitions and controls their execution. This engine includes four modules with different responsibilities. The Policy Server resolves roles and policies, and assigns resources to activities. The Application Server invokes the application programs for each task. The Data Server connects with external data sources such as databases. And the Notification Server handles task notifications to and from worklists.

3.5.2 Internet-mediated workflow

While research on workflow management was making advances on transactional capabilities, object-oriented design, and distributed architectures of workflow systems, the World Wide Web was evolving from static information retrieval to dynamic client/server interaction. From CGI scripts to Web programming languages such as ASP, JSP or PHP, the Web has become the platform of choice for client/server applications. Thus, it became clear that workflow management systems had to take advantage of this new infrastructure. Many commercial workflow products have implemented client/server functionality based on Web technologies, and some workflow research projects have evolved towards these technologies, notably MENTOR (with MENTOR-lite) and METEOR (with WebWork). Recently, other workflow research projects have focused exclusively on Web-based system architectures.

3.5.2.1 Panta Rhei

One of such projects is Panta Rhei [Groiss and Eder, 1997], a workflow management system implemented behind a Web server. Users of this system have access to their worklists through the Web browser, which does not require any special-purpose workflow client application on the user desktop. Panta Rhei also allows tasks to be delivered by e-mail. In this case, the workflow engine sends an HTML form, which the user opens in the Web browser in order to perform the desired task. The form is exactly the same as the user would get from opening the task from the worklist in the previous case. When the task is completed, the user submits the result to the Web server, and the workflow engine will proceed with the following tasks.

Panta Rhei uses a modeling language - Workflow Definition Language (WDL) - that is similar to a script language. Process definitions based on this language are loaded into the workflow engine, which will then implement the corresponding behavior. Panta Rhei allows activities to be assigned to a specific user or to a role. In the first case, the task will appear in that user's worklist. In the second case, the task will be sent to all users belonging to that role. As soon as the first user opens the task for execution, the task is removed from the other worklists.

3.5.2.2 MAGI

In Panta Rhei, when the workflow engine adds and removes tasks to a given worklist, the user will only be able to see those changes the next time the worklist is opened or re-loaded on the Web browser. This situation, that is common in many Web applications, results from the fact that HTTP calls are initiated by the client to the server and there is no event channel from the server to the client. The MAGI project [Bolcer, 2000] has devised an approach to cope with this problem: instead of having one central Web server, every user will have its own Web server. This way, the workflow engine acts as Web client when it sends tasks to users, and users will be clients when they return completed tasks to the workflow engine.

MAGI goes even further and aims at delivering tasks to hand-held devices based on wireless technologies, such as mobile phone and PDAs. However, MAGI realizes that not every user will be able to run a Web server locally. This is especially prohibitive in the case of hand-held devices. Therefore, MAGI has devised a scaled-down version of the Apache Web server, called micro-Apache, with very low memory requirements and an extensible architecture that allows additional modules to be plugged in. The micro-Apache server supports HTTP and other HTTP-based protocols, such as the SWAP workflow protocol described earlier in this chapter. The support for each protocol is encapsulated in a service module that can be loaded or unloaded at any time. External clients are able to determine the supported protocols by using common HTTP methods.

An interesting feature of MAGI is that it allows distributed process execution by sending work directly from one user to the other, without having to submit each task result to a central workflow engine. This is useful, for example, if the purpose of a process is to send a document to be signed by a set of users in a specific sequence. In this case, the MAGI server allows the document and the process definition to migrate from worklist to worklist until the final user returns the signed document back to the workflow engine. OfficeWorks provides this functionality by means of circular e-mail messages. But the advantage of MAGI is that it is possible to determine, at any time, in which worklist the document is, and it is possible to change the course of the document or to change participants in response to exceptions.

3.5.2.3 WW-flow

The WW-flow prototype [Kim et al., 2000] is a Web-based workflow management system that gives special emphasis to the interoperability between workflow engines. WW-flow comprises several components and its functionality can be easily mapped to the Workflow Reference Model, although WW-flow does not comply with WfMC standards. Several user interfaces are accessible through the Web browser, including worklists and administration/monitoring tools. In fact, the WW-flow server has a repository of Java applets that users are able to download to their desktop. Process modeling is the only stand-alone tool, and process definitions are expressed in a proprietary format. Applet-based administration and monitoring tools are able to graphically display process definitions. This same

functionality is available to workflow participants, allowing them to visualize the process concerning a given task.

One of the main features of WW-flow is nested process modeling and execution, which WW-flow calls *run-time encapsulation*. WW-flow allows processes to be defined hierarchically, where each activity may represent an arbitrary sub-process. These sub-processes can then be reused in several process definitions and they are helpful in encapsulating run-time behavior. An activity that represents a sub-process must have its inputs and outputs defined, regardless of how the sub-process is internally structured. This is somewhat similar to the nested graphs of WASA project shown in figure 3.27. In WW-flow, however, an activity may represent a sub-process that is executed externally at another workflow engine. The structure of this sub-process is unknown to the local workflow engine. Using HTTP, a WW-flow server is able to request the remote execution of a sub-process (client behavior) and to receive the results from another workflow engine (server behavior).

3.5.2.4 MILOS

Another Web-based system built on Java technology is the MILOS prototype [Maurer et al., 2000]. MILOS is a tool that provides project planning and workflow management capabilities over the Internet. Its purpose is to support the coordination of software development teams. Project plans are defined using Microsoft Project and an extension to this product that allows information flows to be specified. A project plan is equivalent to a process definition, since it comprises a sequence of steps, each one with a set of input and output documents. A project planner loads project plans into the MILOS server. When plan execution is started, the MILOS server sends e-mails to notify all participants of tasks that have been assigned to them. Users will then log into the MILOS server in order to access their worklists. In the worklist, each task is shown together with its due date and its input and output documents, although input documents will only be available when the preceding tasks have been completed. Whenever that happens, the MILOS server sends an e-mail notifying the availability of the input documents.

An important feature of MILOS is that it supports dynamic process changes. To insert changes, the planner logs into the MILOS server and re-opens the project plan. The new plan is then submitted to the MILOS server, which identifies all changes, analyzes task dependencies and notifies all affected participants that changes have been made to their tasks. This functionality is based on ECA rules: there are rules to enforce the automatic update of affected processes and documents, and rules that specify who should be notified about a certain change. After the MILOS server has determined all task changes, it runs through a list of ECA rules trying to match each change to the event specified in each rule. Whenever a match is found and the rule condition is true, the corresponding rule action is performed. For example, an ECA rule may specify that the event is a change in the task deadline, the condition is the new date being earlier than the old date, and the action is to notify the assigned participant that there is a shorter deadline for the task.

3.5.3 Inter-enterprise workflow

From a technological point of view, the rise of the Internet and its associated technologies has brought new infrastructures - above all the Web and its protocols - that enable the application of workflow management on a virtually unlimited scale. From a business point of view this means that, whereas classic workflow research has devised database-centered approaches or special-purpose infrastructures in order to implement workflow management in an enterprise environment, the new Internet

and Web infrastructures allow the implementation of workflow management either within the same corporate environment, or across enterprise boundaries and into new forms of workflow management, such as inter-enterprise workflow management.

On the other hand, the evolution towards business networks and new organizational paradigms such as the virtual enterprise requires business process management techniques in order to coordinate the activities of several business partners. Given that now there is an infrastructure that enables the use of workflow management on a global scale, this paradigm will become an important ingredient to support the coordination of business processes between enterprises as, indeed, the following research projects show. It should be noted, however, that some of these research projects were not meant to specifically address workflow research issues; rather, they have realized that workflow management was an essential element in their architectures.

3.5.3.1 PRODNET II

The PRODNET II project [Camarinha-Matos and Afsarmanesh, 1999a] aims at the development of an open platform to support virtual enterprises composed of small and medium-sized enterprises (SMEs). PRODNET II has devised a comprehensive infrastructure including support for EDI, STEP, distributed information management, workflow management, virtual enterprise configuration, secure communications, production status inquiries, and quality control. The architecture is depicted in figure 3.29, which illustrates the modules to be installed at each enterprise together with communication with other virtual enterprise members. These modules are divided in three main blocks: the PRODNET Cooperation Layer (PCL), the Advanced Coordination Functionalities, and the existing enterprise systems.

From the enterprise legacy systems, the PRODNET architecture singles out production planning and control (PPC) capabilities, since it considers that order management is one of the most important functionalities to be supported in a virtual enterprise environment [Camarinha-Matos and Afsarmanesh, 1999b]. Hence, existing production planning and control capabilities must be integrated with PCL so that virtual enterprise members can exchange orders electronically. In order to assess the requirements for integrating PPC with PCL in a general case, PRODNET used a commercial production planning system as a test-bed, and has integrated PCL with that system.

The PCL is the main block of the PRODNET architecture since it supports all interactions between virtual enterprise members. The STEP module handles technical product data that is exchanged using STEP standard formats. The purpose of the EDI module is to encode and decode orders and other messages using EDIFACT syntax. EDI messages may contain STEP product specifications. The Configuration Module is used to specify the structure of the virtual enterprise and the interoperability modes (message sequences) for each of its members. These interoperability modes must be taken into account when defining processes for execution at the Workflow Manager.

In fact, the Configuration Module and the User Interface Module provide the Workflow Manager with a Graphical Workflow Editor that is able to create WPDL-compliant process definitions. Processes are defined as flows of control and data between any PCL services, such as those provided by the EDI module, the Communication Infrastructure, the Distributed Information Management System (DIMS), or any other module. For example, a process definition may include a sequence of three activities where the first activity requests the EDI module to create an EDIFACT message, the second activity requests the communication infrastructure to deliver the message, and the third activity requests DIMS to delete some data; in fact, there is a process definition that follows exactly this sequence [Camarinha-Matos and Lima, 1999]. The WPDL process definitions are then loaded

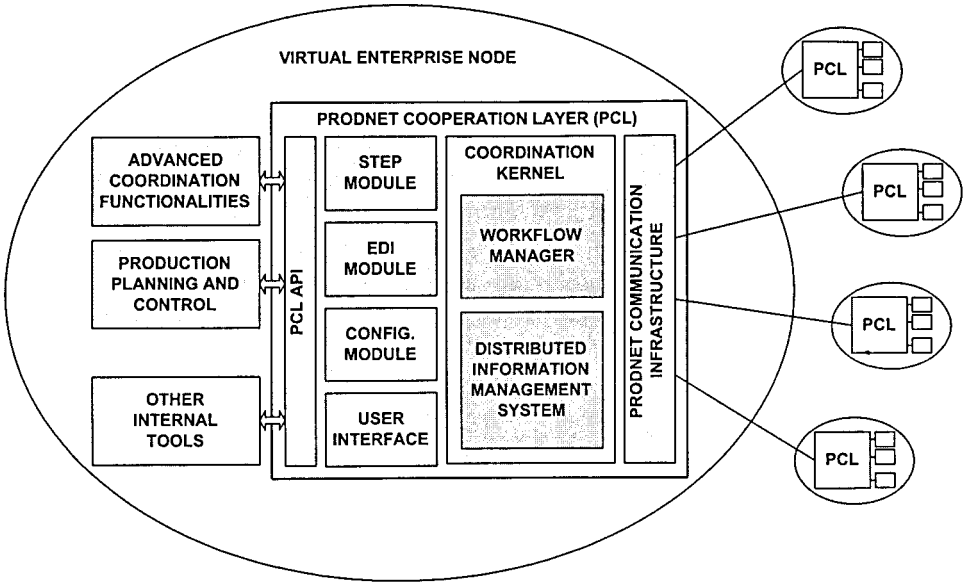


Figure 3.29: Architecture for the PRODNET II project[49]

into the Workflow Manager, which will execute them by sending messages to and receiving messages from other PCL modules. A Coordination Monitor, which is similar in appearance to the Graphical Workflow Editor, allows processes to be monitored at run-time.

The Distributed Information Management System (DIMS) models and manages the exchange of all information within the PRODNET architecture, both with other modules and between virtual enterprise members. DIMS implements a federated database architecture [Afsarmanesh et al., 1999] and it distinguishes local data from imported data and exported (shared) data. All DIMS modules in all virtual enterprise nodes use the same global data schema so that each DIMS is able to retrieve data that is distributed through several other nodes. However, DIMS implements different levels of information visibility that are configured at each node. For example, the information available at one node may not have the same visibility level for all other members. As a result, DIMS supports federated data access only to authorized data at each node. DIMS is able to perform federated query processing on these data.

The PRODNET Communication Infrastructure provides several communication mechanisms between PCL nodes, namely TCP/IP, SMTP/POP3 e-mail protocols, Web-based CGI scripts with security enhancements, and PECP, a proprietary communication protocol [Osório et al., 1999]. The Communication Infrastructure has four main characteristics: communication availability refers to the ability to use alternative protocols to exchange the same information; communication security ensures message privacy, integrity and authentication; communication with legacy systems and other external systems is supported by e-mail protocols; and secure access from a Web client is guaranteed by means of CGI scripts with security enhancements, such as digital certificates and message encryption.

The Advanced Coordination Functionalities comprise two main groups of functionality. The first is connected with supplier search and selection. PRODNET is able to search for suppliers in two kinds of sources: internal suppliers directory and external suppliers directory [Camarinha-Matos and

[49][Camarinha-Matos and Afsarmanesh, 1999b]

Cardoso, 1999]. The internal suppliers directory contains information about the business partners that have had commercial relationships with the enterprise, and this information is assumed to be maintained by the PPC system. It contains product codes used by the supplier and the PPC system, commercial conditions pertaining to those products, and minimum and maximum order size. In addition, it contains a ranking of suppliers according to performance measures such as percentage of rejections of parts, delivery times, and after-sales support.

The external suppliers directory relies on public Internet registries containing lists of companies ordered by industry sector and class of products they supply. PRODNET has identified several of such registries, though each of them has its own data organization and interfacing rules. Thus, PRODNET provides the Electronic Partners Search Tool (EPST) that assists in finding suppliers in three of those registries. The EPST allows the user to specify generic search parameters such as keywords, business division, and location. From these parameters, the EPST generates different queries for the three registries and then collects, parses, and presents the results to the user. These results concern mainly contact information and, in order to obtain further information, PRODNET provides support for an additional step called *call for tenders*. This step generates an HTML/CGI form with a set of questions to be answered by the potential suppliers, and an e-mail is sent to those suppliers inviting them to reply using the generated Web page. The questions on the Web page are chosen so as to collect roughly the same information as that available from known suppliers (through the internal supply directory). Every time a supplier submits the answers to those questions, an e-mail is sent to the original user who initiated the call for tenders.

The second group of the Advanced Coordination Functionalities is provided by the Distributed Business Process Management System (DBPMS) which applies only to the virtual enterprise coordinator, the foremost member in the virtual enterprise structure defined by PRODNET [Camarinha-Matos and Afsarmanesh, 1999c]. The virtual enterprise coordinator periodically receives a set of information items from each member, such as order status, remaining process time, or the amount of parts already produced [Klen et al., 1999]; this information concerns mainly the status of processes running in each member. The set of information items to be provided by each member to the virtual enterprise coordinator is specified in "supervision clauses". The whole functionality of DBPMS is based on these supervision clauses. The DBPMS continuously monitors received information in order to detect any deviation or conflict according to the supervision clauses. Such conflict may be, for example, the delivery date for the order of one member becoming later than another member requires. In this case, DBPMS is able to provide the user with alternative solutions, based on several rescheduling options.

3.5.3.2 VEGA

The VEGA project [Zarli and Poyet, 1999] aims at establishing an IT infrastructure which supports the technical and business operations of virtual enterprises. According to VEGA, such an infrastructure must cope with several requirements, namely (1) the exchange of product information between software applications across the virtual enterprise, (2) information consistency and controlled concurrent access to that information, and (3) flexibility and scalability of the underlying IT infrastructure. To cope with these requirements, VEGA has realized the need for the following key technologies: product data modeling (using STEP and other standards), middleware technology (using CORBA), workflow management (using mainly commercial products), and the Web and its associated technologies (in particular HTML, CGI, and VRML). The VEGA project architecture comprises four components that employ those key technologies:

- The first component, the COAST communication platform, relies on CORBA in order to enable access to distributed product data. From the services provided by this communication platform, some are based on standard OMG services, while others were developed purposely for COAST.

- The second component is the Distributed Workflow Service (DWS), which is able to distinguish between process activities that are public to other virtual enterprise members, and those activities that are internal to the enterprise and should be kept private. The DWS specification requires both interoperability with other workflow management systems and global workflow monitoring. The implementation of DWS is based on a commercial workflow product that handles the global (public) virtual enterprise processes. This global workflow system is able to invoke applications directly and to launch activities on local workflow systems that belong to virtual enterprise members. The VEGA project has demonstrated this possibility by integrating DWS with another commercial workflow tool. In essence, the DWS is responsible for global virtual enterprise processes, while private processes are assumed to be handled by local workflow systems in each member. In addition to DWS, VEGA has developed a workflow process meta-model that links product data with public process definitions. This meta-model supports the creation of public process definitions according to WfMC's WPDL.

- The third VEGA component is concerned with the interoperability of STEP product data between different applications across the virtual enterprise. The Schema Interoperability Service (SIS), as it is called, provides file input/output, conformance checking and schema mapping for product data based on the STEP language.

- The fourth VEGA component is the Distributed Information Service (DIS) and its purpose is to access distributed information through COAST and present it at user desktops by means of Web front-ends. In addition, Web-based, user-oriented services include STEP-compliance checking of product data, visualization of geometrical STEP data using VRML, and analysis of product data structure.

These functionalities, and their integration with other VEGA components such as the DWS and COAST, are implemented using Java applets and CGI scripts. For example, when the user opens the Web browser, an applet is downloaded and the user logs into the DWS. The applet then shows the worklist for that user and, for each task, the browser retrieves product data from a local database through a CGI script that connects to the COAST communication platform.

3.5.3.3 PLENT

The goal of the PLENT project [Ktenidis and Paraskevopoulos, 1999] was to develop a set of innovative software tools to support coordinated production planning and control in a network of manufacturing SMEs. For this purpose, PLENT devised three software modules: the Coordination Unit (CU) module, the Node Manager (NM) module, and the Workflow Manager (WfM) module. The CU module is responsible for the interaction with the final customer, selection of virtual enterprise members (called nodes), and assignment of tasks to those nodes. The NM module, to be installed at each node, supplies the CU module with delivery plans and conditions. The WfM module manages all message exchange between virtual enterprise members in order to synchronize the behavior of those entities.

The PLENT project has developed its own process modeling constructs that can be used to describe the virtual enterprise using a Network Operational Schema (NOS). The NOS is a product-oriented process model defined at the network level. It does not delve into the process details of each

member, but rather represents only macro-activities at each node and transportation steps between nodes. The NOS includes information about production capacities, manufacturing and transportation times. It represents the basic information structure needed to plan and coordinate the virtual enterprise. The CU module provides tools for creating, editing and analyzing the NOS.

From the data represented in the NOS, the message exchange requirements will define workflow behavior. All outgoing messages from each node are sent to the WfM module, which performs any branching decisions and sends the messages to the appropriate recipients. In fact, process instances are created in response to received messages. All exchanged information is stored in a dedicated archive, which is controlled by the WfM module. The CU module has read-only access to this message archive. On the other hand, the WfM module has read-only access to the NOS. The WfM module and its message archive have been implemented in a Web server. Hence, a message is not physically sent to a virtual enterprise member; instead, the member must open the message from the Web server, using a special-purpose Web page. In a similar way, when a node wants to send a message, it must insert the message in the archive by means of another Web page.

3.5.3.4 ACE-Flow

The ACE-Flow project [Stricker et al., 2000] does not propose a workflow system architecture, but rather an electronic trading system for business processes. ACE-Flow realizes that there is a general trend towards outsourcing and has devised a trading system for the enterprise to subcontract services. Internal enterprise processes, together with outsourced services, can be considered to be inter-enterprise business processes. ACE-Flow assumes that each enterprise has its own workflow management system to manage its own processes and, in order to coordinate internal processes with outsourced processes, the trading system attempts to federate workflow management systems from different enterprises.

The ACE-Flow trading system is an external system with two databases. One database is the service catalog, i.e., the set of services publicly available to enterprises. Each of these services is, in fact, an internal process from a certain enterprise. In the trading system, the process is listed as a service with an identifier, a description, a set of formal input and output parameter definitions, a simplified state transition diagram, and a set of *facets*. The state transition diagram defines only a set of *milestones* that are used to track the progress of the subcontracted service at run-time. Facets describe relevant service characteristics, such as process length or execution costs, that can be used to compare and select services based on given criteria. For example, an enterprise may be willing to contract a verification service, with certain maximum values for cost and length, to be performed at a nearby location.

The second database contains workflow data that concern the execution of inter-enterprise processes. This execution is carried out according to a pre-defined procedure. At first, an enterprise workflow system sends a request to the trading system while looking for a certain service. The request includes a reference to the desired service, chosen from the types of service available in the electronic catalog. It also includes a set of facets and the corresponding values that the service should comply with. Based on the request and on information from the electronic catalog, the trading system tries to determine the eligible service providers. If some of the requested facets are not specified in the available service definitions, the trading system will ask each eligible provider for a bid to execute the service, taking those facets into account.

The workflow system from each of those service providers will then return a bid with specific values for those facets. The trading system determines the bid that fulfils the original request or, if

more than one bid is possible, it inquires the contracting enterprise to select the desired bid. Once the bid is chosen, the trading system requests the service execution from the respective provider. During execution, the service provider keeps the enterprise aware, via the trading system, of the reached milestones. When the service is completed, the results are transferred to the enterprise.

3.5.3.5 WHALES

The WHALES project [Gazzotti, 2001] places its focus on enterprise services and on the infrastructure support for linking these services, giving special emphasis to Web technologies. The purpose of WHALES is to provide a planning and management infrastructure for distributed organizations working on large-scale engineering projects. According to WHALES, such processes are lengthy, resource-intensive, and include disparate activities with temporal constraints, both on milestones and process duration. As a result, these large-scale engineering projects often result in a distributed organization with a high uncertainty about budgets and profit margins. WHALES aims at more reliable project plans and budgets by harmonizing data from heterogeneous applications and different domains, while improving information flow, notification of events, and re-planning capabilities across companies.

WHALES aims at producing unified and generalized process and data models that can be used in different industry and service sectors, and it aims at supporting distributed organizations by allowing links (and roles) between companies to be configured. WHALES has devised a common Web-based working environment that integrates data from different business units, enforces data access rights based on roles, and provides workflow capabilities. Through integration with local applications at each enterprise, WHALES aims at supporting the whole engineering project life cycle. WHALES emphasizes on workflow management as one of its key contributions to the coordination of inter-enterprise business processes throughout the project life cycle. The other key contribution is a Web-based project environment as the underlying infrastructure for interaction with local and remote users.

In WHALES, process models are defined according to three layers. The first layer is the Work Breakdown Structure, which is basically the sequence of activities that takes place for a given business process. Within this layer, process models are similar in appearance to common process definitions, except for the fact that they do not have any reference to assignments or to resources that perform individual activities. These issues are addressed by the second layer, the Work Accountability Structure, which refers to the assigned resources. The third layer, the Work Network Structure, specifies the enterprises that the resources belong to, and the organizational relationships between resources and between enterprises. This third layer resembles an organizational model of the distributed organization and of its constituent enterprises, relating the resources that participate in a business processes during the project life cycle.

3.5.3.6 WISE

The WISE project [Lazcano et al., 2000], like WHALES, has realized the need for a software platform that integrates enterprise services over the Internet. WISE is strongly focused on providing workflow management support for inter-enterprise business processes, and its architecture includes a set of modules that support process definition, enactment, monitoring, and coordination. However, instead of focusing on a single workflow system with all these features, WISE tries to combine different tools in order to cover that functionality, and to come up with a software platform for the virtual enterprise. Given that each enterprise has its own internal processes, the purpose of WISE is to facilitate the

incorporation of information and communication technology into business processes so as to expand them beyond corporate boundaries. Business processes then become virtual business processes.

In a virtual business process, individual steps correspond to sub-processes that take place at different organizations. Its building blocks are the internal processes from each enterprise, although full details about these internal processes may not be known. Knowing their inputs and outputs is enough to be able to define virtual business processes. WISE defines a virtual enterprise as an organization that is based on virtual business processes. The set of members that participate in a virtual enterprise is called the trading community. The trading community is the set of enterprises that provide the building blocks for virtual business processes. Each virtual enterprise has one trading community and possibly several virtual business processes.

Figure 3.30 illustrates the relationships between these concepts. The relationships shown represent what WISE calls the *virtual business process life cycle*. The life cycle begins at the trading community, when companies advertise their services and other companies incorporate those services into their own business processes, so as to define a virtual business process. As in ACE-Flow, the services are available from an online catalog. The second phase is the enactment of that virtual business process across the Internet. This enactment is equivalent to the operation of the virtual enterprise. During the third phase, the virtual enterprise monitors the executing processes and analyzes their behavior and performance in order to continuously improve those processes. In the fourth phase, and in parallel with the third phase, partners in the trading community exchange information with each other according to their responsibilities within the virtual business process. The communication channel is established not between two partners, but rather between the workflow system and each virtual enterprise member, based on the dynamics of process execution.

The WISE architecture is a combination of tools that provide support to these four phases. Process definition is supported by a Web catalog of services and by a commercial modeling tool. The Web catalog uses Java applets and servlets to allow companies in the trading community to advertise their services and to find services provided by other companies. It shows the semantics for each service and it contains objects that encapsulate service behavior. The virtual business process integrates these services by establishing the control flow and the data flow between them. For this purpose, a commercial modeling tool called IvyFrame[51] is used, which is internally based on Petri nets. With this tool, the user is able to build the virtual business process definition in a drag-and-drop manner

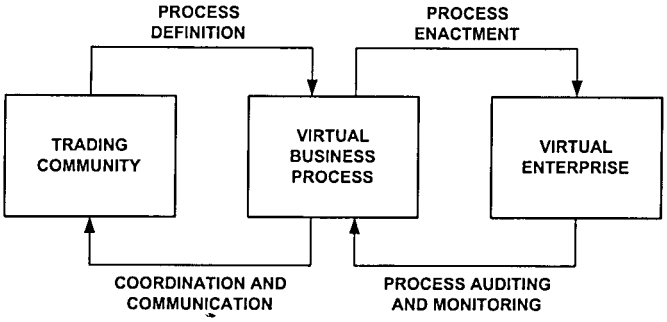


Figure 3.30: Key concepts in WISE[50]

[50] [Lazcano et al., 2000]
[51] http://www.ivyteam.com/

from the Web catalog. Moreover, the user is able to specify its own enterprise services, and IvyFrame automatically generates the encapsulating code for the service and inserts it in the catalog.

Some steps from the virtual business process may be defined not as external services, but as invocation of internal applications, such as invoking a local ERP function or a local workflow management system. In fact, WISE has been integrated with SAP R/3 Business Workflow and IBM FlowMark [Schuler et al., 1999]. Therefore, a virtual business process definition may combine external services and local tasks. WISE also realizes that most companies may already be using their own modeling tools and, as a result, WISE aims at compiling existing modeling languages into a language called Opera Canonical Representation (OCR), which comes from a process support system called OPERA [Hagen and Alonso, 1998]. However, the user has no contact with the OCR language, because IvyFrame is used as the modeling front-end. The OCR representation, to be used internally, is a complete description of the virtual business process, containing the specification of the sequence of steps and the applications or services they invoke from each enterprise.

The second phase, process enactment, is supported by a workflow engine that has been built as an extension to the OPERA system. The engine interprets the OCR process definitions and coordinates the flow of information between information systems in the trading community. Inside the engine, every step of execution is recorded in the system database for monitoring and recovery purposes. To be able to interact with other systems, the workflow engine uses the concept of *subsystem adapters* that encapsulate the interaction with external systems. These adapters can be specified and automatically created by means of a graphical user interface, which allows new components to be easily added to the system. These and other graphical user interfaces are built using Java technology.

WISE emphasizes three main features of its workflow engine. One is security, since the engine employs a set of security mechanisms including encryption of data before sending it over the network, and user authentication for accessing process execution and monitoring. Another feature is Quality of Service (QoS): based on execution statistics and network characteristics, the workflow engine is able to predict the usage of network resources and to reserve those resources through a specific resource reservation protocol. The third feature is fault-tolerance, which WISE implements using transactional concepts. WISE separates the ACID properties into three constructs that are used to group process activities according to the desired semantics: WISE uses *spheres of atomicity* to determine whether process execution will always reach a consistent point or if there are irrecoverable execution paths, *spheres of isolation* are used to synchronize access to shared resources, implementing concurrency control, and *spheres of persistence* are used to define which activities will have their results permanently stored (by default, all activities are persistent).

Process auditing and monitoring is again supported by the IvyFrame modeling tool. At any time during execution, IvyFrame allows processes to be exported as Java applets which display the process model as well as its current execution status. IvyFrame provides the user with additional administrative capabilities such as creating process instances, and determining the set of process instances running in the virtual enterprise. Besides, the workflow engine maintains a history space where information about all completed processes is stored and organized in order to facilitate its analysis. This analysis serves two purposes: one is to support decision making regarding the configuration and status of the system, and the other is to collect information about configuration, resource reservation, and load balancing, that can be used in real-time to support QoS features.

Regarding the fourth phase, coordination and communication, WISE realizes that partners in a virtual enterprise environment must be able to solve unavoidable inconsistencies or minor problems during process execution. This requires collaboration between users at different enterprises. In order to support this collaboration, WISE employs results from a previous project called CoBrow [Sidler et

al., 1997], which has developed mechanisms for interactive and non-interactive collaboration over the Web. These collaboration mechanisms include multimedia capabilities such as video-conferencing, which the user can invoke from the Web catalog directly, so as to contact the partner responsible for performing a certain service within the virtual business process.

3.5.3.7 COSMOS

Several research project architectures, such as ACE-Flow and WISE, do include a service catalog where services provided by other enterprises can be found. In the COSMOS project [Weinberg, 2000], however, that catalog does not describe services, but rather it contains templates for electronic contracts. An electronic contract can be achieved by negotiation between two enterprises, and it describes the inter-enterprise business process that will take place between those entities. The COSMOS project has devised a comprehensive architecture with a strong focus on supporting electronic contracts. Its central idea is to establish an electronic marketplace where electronic contracts are negotiated, electronically signed, and executed. In this architecture, particularly during contract execution, workflow management plays an important role.

According to COSMOS, trading between enterprises follows a four-phase life cycle. The first phase is the e-marketplace, when each enterprise places offers or requests for certain services. These offers and requests are expressed as a contract template: a document that partly specifies how the service is performed and what are the requirements that the contracting enterprise should comply with. The contract templates are stored in a catalog where they can be found by other enterprises. Once that happens, the second phase, called contract negotiation, begins. During contract negotiation, both parties extend and modify the original contract template so as to come up with an electronic contract that satisfies them both. After they reach an agreement, they digitally sign the contract - this third phase is called electronic signing. Furthermore, the contract is signed by an electronic notary that countersigns and authenticates the contract. The contract contains the process definition to be used during the fourth phase, called contract performance.

COSMOS distinguishes between three types of contracts. One is the bilateral contract just described, which takes place between two parties. A second type is a consortia contract, which is signed by all members of a consortia such as a virtual enterprise. The third type is the multi-layer contract, which applies to several layers of subcontracting. For example, a company may contract the service of another company which, in turn, subcontracts several others. In this case, the top-layer contract is only signed when the lower-level contracts have already been signed with all subcontractors. As a consequence, COSMOS supports two kinds of business networking: either consortia made up of several enterprises, or relationships across several layers of subcontracting.

In both cases, an electronic contract undergoes three stages. The first stage is information, when the contracting parties get to know each other through the catalog. After a contract template has been chosen from the catalog, the negotiation stage begins. During this stage, the contract is modified until all parties agree and sign it. Then the execution stage tracks the duties of each party according to the signed contract. The COSMOS architecture includes functionality that supports these three stages. In the information stage, the catalog plays the main role. In the negotiation stage, the contract editor is the most important component. And during the execution stage, a workflow management system supports contract performance. This architecture is shown in figure 3.31; all components have been implemented using Java technology.

One of the central components in the COSMOS architecture is the Contract API, which connects most of the remaining components. The Contract API accesses the catalog and contract databases

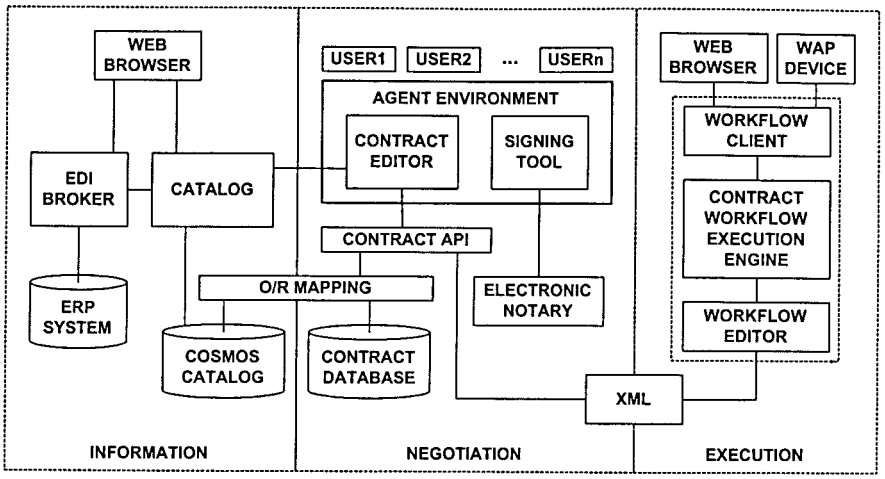


Figure 3.31: Architecture for the COSMOS project[52]

through an object/relational mapping, and exchanges contracts with the workflow management system in an XML format. The electronic contracts are produced during the negotiation stage, which is supported by a contract editor. The contract editor can also be used to create new contract templates to be stored in the catalog. The catalog belongs to the information stage and is accessible either through a special-purpose Web interface or through an EDI interface. In the negotiation stage, the contract editor and the signing tool, which allows an electronic notary to countersign contracts, work in a agent-based environment. Within this environment, mobile agents carry the contract between parties (users in figure 3.31) as many times as necessary until the contract is signed.

To support contract execution, COSMOS provides three workflow components: the Workflow Editor, the Contract Workflow execution engine, and the Workflow Client application. The Workflow Editor is a modeling tool based on the Unified Modeling Language (UML) [Booch et al., 1999]. Using this language, processes can be defined as UML Activity Diagrams, which include familiar modeling constructs. In addition, COSMOS uses constructs for explicit and implicit choice. Basically, an explicit choice is a branching decision performed by the workflow system based on available variables or conditions; an implicit choice is a branching decision to be performed by the assigned user, i.e., the user decides which activity to perform. The Workflow Editor also makes use of several other extensions to UML Activity Diagrams. For example, it has actors associated with activities, it models information objects, and it has special constructs for queries and automatic tasks (application invocations). The resulting process definitions are exported in an XML format so that they can be included as part of the electronic contract, and signed together with the contract.

The Contract Workflow execution engine compiles the Activity Diagrams from the Workflow Editor into a special kind of Petri nets called *reference nets* [Aalst et al., 1999]. Reference nets extend Petri nets by introducing the concepts of net instances, references between net instances (hence the name "reference nets"), and synchronous channels to communicate between net instances. This allows the Contract Workflow execution engine to create process instances, to exchange data between process instances, and to choose sub-processes at run-time. The Contract Workflow execution engine relies on Renew [Aalst et al., 1999], a simulator that runs reference nets. Renew includes a client

[52] [Weinberg, 2000]

application that was also adopted in COSMOS, and was the basis for the Workflow Client application. The Renew server and client are built on Java technology and communicate via Remote Method Invocation (RMI)[53]. In addition, the Workflow Client application employs the results from a project called Hydepark [Müller-Wilken, 2000] in order to allow access from mobile devices.

3.5.3.8 CrossFlow

Another project that focuses on electronic contracts and inter-enterprise workflow management is the CrossFlow project [Grefen et al., 2000]. CrossFlow argues that virtual enterprises can only be successful if cooperation between their members can be set up in a dynamic and flexible manner. CrossFlow regards virtual enterprises as enclosing business processes that are linked across organizations. These business processes represent business relationships of provision and consumption of services between the members of a virtual enterprise. Therefore, CrossFlow realizes the need for automated ways for finding, purchasing, integrating and managing services performed by different enterprises. Furthermore, it realizes that connecting the business processes of different enterprises is a prerequisite for virtual enterprises. It is thus the goal of CrossFlow to support the integration of business processes across organizational boundaries.

In order to address this requirement, CrossFlow attempts to link the workflow management systems of different enterprises. However, CrossFlow argues that current workflow management systems cannot address this challenge, and that the interoperability of workflow systems, although necessary, is not enough to connect those business processes. The approach proposed by CrossFlow covers instead two fundamental issues. The first is an *end-to-end view* of the cross-organizational business processes, of its constituent business processes at each enterprise, and of the transformations required to link those processes across enterprise boundaries. The second issue is the *business life cycle*, which describes the steps towards initiating contact between enterprises, reaching an agreement, and signing a contract. The business life cycle should also address the configuration of the enterprise information systems, and how to implement (or enact) the contract they have signed.

CrossFlow defines outsourcing as the delegation of part of a business process from one enterprise to another, where the delegation is supported by a legal binding agreement - a contract. A cross-organizational workflow takes place when the outsourced business process is implemented as a sequence of activities at a remote enterprise. The set of activities that are carried out by one enterprise on behalf of another is called a *service*. The service contract, or simply contract, contains a description of the service, as well as the promises made by one party to the other regarding the outsourced business process. The contract defines the business process that represents the provision and consumption of the service. It describes all possible interaction between the two enterprises. The enactment of the service contract is the implementation of the outsourced service, as agreed in the contract.

As depicted in figure 3.32, the workflow management system is at the core of the business relationship between enterprises, because it implements the service being outsourced. In addition to the local workflow system, CrossFlow recognizes that the configuration of internal resources concerns other components as well. For example, it concerns administrative services, which are needed in conjunction with the core service. It concerns authority, i.e., the responsibility of each enterprise to develop and enact a contract as it sees fit, depending on organizational policies and available resources. It concerns business process management, which is responsible for core services and administrative

[53] http://www.java.sun.com/products/jdk/rmi/

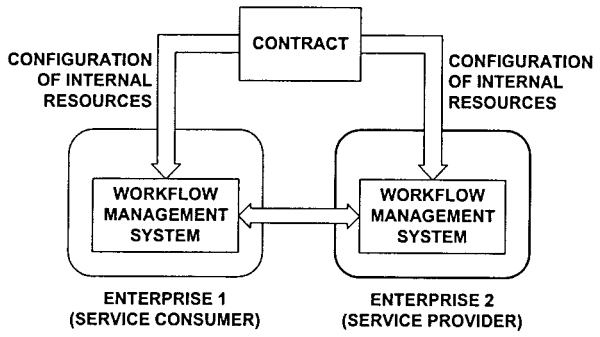


Figure 3.32: The contract as the means to configure internal resources in CrossFlow[54]

processes, and for managing these services according to those policies and resources. And it concerns the configuration of a distributed platform that supports interaction within and between enterprises.

In practice, these elements may work differently from enterprise to enterprise. In particular, workflow management systems have different ways of managing the sequence of work activities, so even the same business process could be represented differently according to each workflow system. CrossFlow copes with these differences in two ways. Whenever possible, standards should be employed in order to minimize the need for data transformations between systems. Such standards are languages to describe products and services, such as STEP, or standardized workflow interfaces such as those proposed by the WfMC and the OMG. In fact, CrossFlow implements a subset of the OMG Workflow Management Facility standard.

The other way CrossFlow copes with mismatches between enterprises is by introducing the concepts of *domain* and *proxy-gateway* between domains. A domain is a homogeneous environment where no data transformations are necessary: each enterprise may be a domain, or it may contain several domains. To overcome the difference in domains from one enterprise to the other, a proxy-gateway is used as an interceptor that performs the necessary data translations. Additionally, the proxy-gateway hides the internal details from each domain, it checks the cross-organizational interactions in order to protect the domain's integrity, and it performs monitoring in order to keep an audit trail of all interactions.

Contract establishment CrossFlow identifies several phases for the business life cycle between enterprises. The first phase is contract establishment and it comprises several steps. The first step is initial contact between enterprises. This contact is motivated when there is a match between what one enterprise offers and what the other needs. In CrossFlow, this match is identified by a matchmaking facility. In a second step, both parties exchange information in more detail about the services being offered and requested. The third step is negotiation, which takes place over service attributes. If successful, both parties reach an agreement. A final step is signing the contract that describes that agreement.

CrossFlow supports the contract establishment phase, but with some simplifying assumptions. One assumption is that the virtual enterprises are created in restricted marketplaces, i.e., marketplaces where there are already contract templates available. This avoids much of the complexity of reaching an agreement, since both parties start by agreeing on the contract template to be used as the

[54]adapted from [Hoffner, 1999]

basis for their service provision-consumption arrangement. Another assumption is that negotiation is outside the scope of the project, since CrossFlow realizes that the automation of negotiations is a complex and open research area. Contract signing is also left out, as it is presumed to be implemented with auxiliary notary services and digital signatures. CrossFlow has, however, developed a contract framework to support the description of contracts and contract matchmaking. The contract framework includes a conceptual contract model, an XML-based contract language, and a contract matchmaking facility.

In addition, and due to the first assumption, the contract framework introduces the concept of contract template. A contract template has the same structure as a contract but with placeholders for specific information to be filled in. A contract instance, simply referred to as a contract, is the result of filling the placeholder fields on a contract template. For example, there may be contract templates for sales and order contracts, bank loans and mortgage contracts, or employment and membership contracts. However, when service provider and service consumer search for each other, contract matchmaking is only possible when both parties use the same contract template. So the first step is to choose the appropriate contract template from the ones available in the marketplace. According to CrossFlow, the service provider uses the contract template to produce a Contract Advertising Template (CAT), which describes the service it implements. On the other hand, the service consumer uses the contract template to produce a Contract Search Template (CST), which describes its need for a certain service. Both the CAT and the CST are submitted to the matchmaking facility. Figure 3.33 illustrates this procedure.

The matchmaking facility is implemented as an extended version of the CORBA Trading Object Service [OMG, 2000c]. Basically, this service has a central trader where service providers advertise their offers and service consumers search those offers according to their needs. The CrossFlow matchmaking facility has a central trader too, but it differs from the CORBA Trading Service because it supports a symmetric matchmaking process, i.e., both consumers and providers describe what they offer and what they require from each other. The requirements from both parties are expressed in a property-constraint language, where properties are name-value pairs and constrains are logical expres-

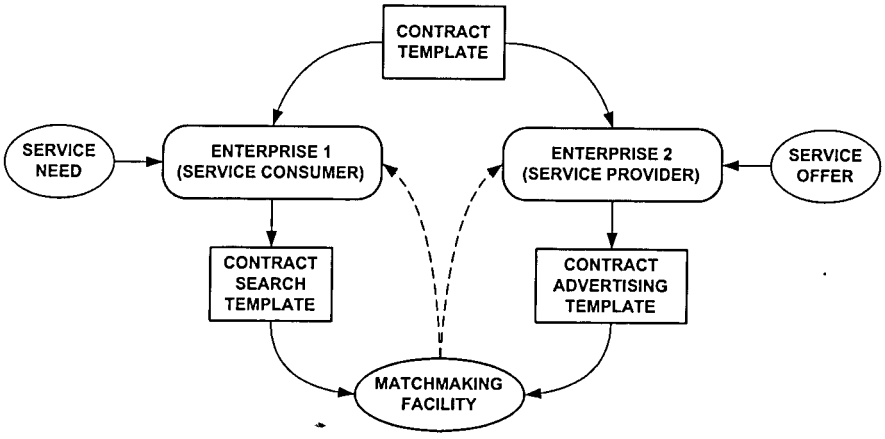


Figure 3.33: Contract templates and the matchmaking facility in CrossFlow[55]

[55]adapted from [Hoffner, 1999]

sions based on those properties. For this purpose, the matchmaking facility includes a data dictionary that defines the properties to be used in contract templates.

Contract enactment configuration The second business life cycle phase is contract enactment configuration, and its purpose is to prepare internal resources for contract enactment. Two inputs are required in this phase: the contract and the Internal Enactment Specification (IES). The IES specifies how the contract is to be implemented within the enterprise, together with several implementation details. It includes the internal resources to be used, the relationship between the service described in the contract and its internal workflow implementation, and the implementation details for the interaction between the service provider and the service consumer. The IES uses an XML-based language, it refers to the proxy-gateway to be used, and to the inbound and outbound operations invoked during the interaction between service provider and service consumer.

Service enactment The third business life cycle phase is service enactment, which represents the actual provision and consumption of a core service. In order to execute business processes that span across different enterprises, CrossFlow has devised a set of services with additional functionality beyond the typical capabilities of workflow management systems. These services are called Co-operative Support Services (CSS). One of such services is Level of Control (LoC), which provides cross-organizational transaction management (allowing rollback of business processes) and cross-organizational process control support (which includes primitives to suspend, cancel, and rollback processes). Another CSS service is Quality of Service (QoS) Monitoring, which monitors user-defined parameters during process execution, and also collects information for offline analysis. The third service is Flexible Change Control (FCC), which provides modeling constructs to support flexible execution alternatives.

The LoC cooperative support service uses a transaction model that distinguishes between three levels of transactional semantics. These levels are the outsourcing level, the external level, and the internal level. The external process level concerns the business process as it is described in the contract. The outsourcing level concerns the business process as it is defined at the service consumer side, i.e., the internal business process together with the outsourced activities. The internal level concerns the implementation, at the service provider side, of the outsourced business process, i.e., it describes the internal activities that implement the business process specified in the contract. At each transactional level, the process definition may include compensating activities that semantically undo completed activities. When rollback is deemed necessary, a compensating process instance is created from the pre-defined compensation activities. This transaction model is similar to those of EXOTICA and WIDE. The LoC cooperative support service includes control operations to stop and resume, to cancel, to rollback, and to change variables of a process instance.

The QoS Monitoring cooperative support service performs two kinds of monitoring: online and offline monitoring. Online monitoring takes place during process execution, when the service consumer is to be notified about the execution of the outsourced service. The service consumer sees the outsourced service as a set of one or more activities, each having one of three possible states: not active, active, or complete. Whenever the state of such activities changes, an event is generated and the service consumer is notified about the new state. These are called observable events: they define the state transitions which can be observed by the service consumer, either by explicit request or by notification. Besides observable events, user-defined parameters can also be monitored, which are equivalent to observable process variables. Based on these parameters, CrossFlow allows contracts

to specify ECA rules for notification: when a certain event occurs, a condition based on parameter values is evaluated and, if true, then a notification is sent to the service consumer.

Offline QoS Monitoring deals with the analysis of information collected during run-time. The QoS Monitoring cooperative support service maintains a log containing the start and end events concerning the outsourced service activities. Each of these events is recorded together with a time stamp. From this log, QoS Monitoring builds a stochastic model based on continuous-time Markov chains. This model shows the sequence and average times for several execution runs of the same service, and it is useful to characterize and predict the expected time and sequence of activities for the outsourced service. This technique is especially useful to derive process models of external services from their externally observable behavior [Klingemann et al., 1999].

The purpose of the Flexible Change Control (FCC) cooperative control service is to support flexible workflow execution. Besides typical workflow modeling constructs such as splits and joins, FCC introduces modeling constructs for execution alternatives. These modeling constructs are called flexible elements, and there are three kinds of these elements. The first kind of flexible element is *alternative activities*: it specifies two possible activities and one of these will be chosen at run-time, based on arbitrary run-time conditions. The second kind is *non-vital activities*, i.e., activities that can be included or omitted during process execution. The third kind of flexible element is *optional execution order*: this construct specifies a set of activities that can be executed in any order, including sequential or parallel execution. An interesting point is that the execution of flexible elements, and the run-time decisions they require, can be made based on QoS parameters. In this case, the FCC module chooses the execution options that optimize the weighted sum of QoS parameters.

The Workflow Module The three cooperative support services (LoC, QoS Monitoring, and FCC) extend the functionality of the local workflow management system. In the CrossFlow architecture the interface between cooperative support services and the local workflow system is the Workflow Module (WM). The Workflow Module implements a vendor-independent, Java-based interface, which is a subset of the OMG Workflow Management Facility standard. The Workflow Module, however, extends this interface in order to support advanced monitoring and control functionality implemented by the QoS and LoC cooperative services. Additionally, the Workflow Module replicates process state information to the service consumer's side. As a result, the Workflow Module at the service consumer behaves as if it would be a local performer of the outsourced service.

In order to demonstrate this concept, CrossFlow uses IBM MQSeries Workflow as the local workflow system. In fact, this system already provides a Java-based API (called JAPI) for workflow client applications. However, JAPI is unable to notify external programs about changes occurring in the workflow system. Workflow client applications must poll the workflow system to get such information. JAPI can thus be referred to as a pull interface. One of the main features of the Workflow Module is that it provides a push interface, i.e., the Workflow Module is able to notify other modules (such as CSS) about events that occur in the local workflow system. Furthermore, the Workflow Module is able to notify other Workflow Modules at remote sites about local workflow events. This is especially useful for a service provider to convey observable events to the service consumer.

The CRAFT integration infrastructure According to the CrossFlow architecture, communication between enterprises is achieved through the CRAFT integration infrastructure. The CRAFT integration infrastructure is comprised of a set of CRAFT integration facilitators at each enterprise, where each CRAFT integration facilitator supports exactly one contract. The CRAFT integration facilitator

is the entry point for all incoming interactions and the exit point for all outgoing interactions concerning a specific contract. In practice, and according to CrossFlow, an interaction is just a remote function call between two CRAFT integration facilitators at different sites. In fact, CRAFT integration facilitators are implemented in Java and communication between these modules is attained using Remote Method Invocation (RMI).

Each CRAFT integration facilitator is composed of four types of components, as shown in figure 3.34. The first component is the Contract Manager, which is used to enable and disable interactions concerning a specific contract. For example, an outsourced service may be started locally or by a remote request, but this is only effective when the corresponding contract has been enabled by the Contract Manager. The second component is the proxy-gateway (PG), which is a communication gateway to the outside. Theoretically, a proxy-gateway can support several transport format and protocols, but in CrossFlow inbound and outbound interactions have been implemented using RMI. The third type of component is the Functional Extension (FX), which deals with incoming interactions and requests outgoing interactions. Functional Extensions connect to the local Workflow Module and possibly to other back-end systems as well. They deliver inbound interactions to these systems, and accept requests for outbound interactions from these systems.

The fourth component is the Coordinator, which is the central component of the CRAFT integration facilitator. The Coordinator knows which Functional Extensions to notify when an inbound interaction arrives, and it distinguishes between Functional Extensions that are *actors*, *supervisors*, and *listeners*. An FX is an actor if, for an inbound interaction, it can influence the behavior of service execution such as creating a local process instance or finishing a running process. Supervisor functional extensions decide whether an inbound interaction is valid, and whether it can be dealt with according to the current service execution state. They also decide whether the request for an outbound interaction should be granted or not. An FX is a listener if it receives notifications about inbound and outbound interactions, but it cannot influence service execution.

At run-time, and after the contract has been enabled through the Contract Manager, inbound interactions are dealt with in four steps. In the first step - authorization - the Coordinator calls all supervisor Functional Extensions and determines if the interaction can be admitted for processing. If all supervising FXs agree, the Coordinator proceeds to the second step, otherwise the interaction is returned back to its sender. The second step is pre-operation notification, and during this step the Coordinator notifies all listener FXs about the inbound interaction. In the third step - performance -

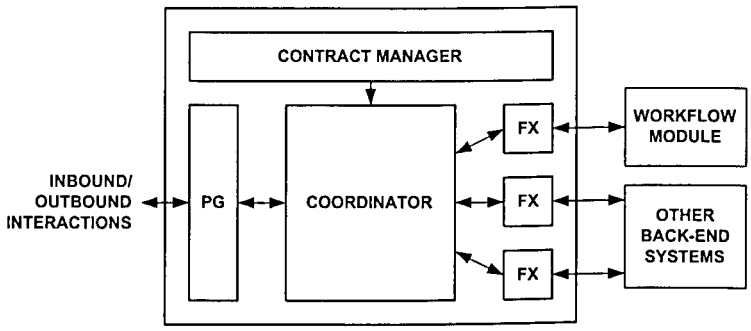


Figure 3.34: Components of the CRAFT integration facilitator in CrossFlow[56]

[56][Saint-Blancat, 2000]

the Coordinator sends the interaction to the corresponding actor FX, which delivers it to the Workflow Module or other service execution system, and waits for a result. The fourth step is post-operation notification: listener FXs are notified about the completion of interaction processing. When the fourth step is complete, the result is sent back as an outbound interaction to the original sender of the inbound interaction.

3.6 Concluding remarks

From concepts to standards and from commercial products to research projects, this chapter has presented an overview of significant developments in the field of workflow management systems. Whereas the first research efforts in this field were strongly connected with database technology, today workflow management focuses on the coordination of business processes that extend beyond enterprise boundaries. If the previous chapter has shown that enterprises are evolving towards business networks, then this chapter shows that workflow management systems have both the aptitude and potential to support the coordination of business processes in an inter-enterprise environment. The key enabling factor is, like in the previous chapter, the introduction of the Internet and its associated technologies. Whereas in the previous chapter these have been shown to facilitate new business structures and relationships, in this chapter it was shown that these technologies have fostered considerable advances in the capabilities and applications of workflow management systems.

Most commercial workflow systems are now able to deliver tasks through Web pages and e-mail, and some of them have been implemented according to Web-based, client/server architectures. At the same time, standardization efforts have realized the importance of distributed object systems and Web protocols, so that bindings of existing standards to the these technologies, as well as new standards based exclusively on Web protocols, are being developed. The research arena is where most new technologies have been experimented with, such as transaction processing monitors, distributed object-oriented architectures, and XML-based data formats. Research initiatives were also the first to realize and demonstrate the importance and potential of workflow management regarding the integration of business processes between enterprises. In fact, as this chapter has shown, and figure 3.35 illustrates, workflow research projects have addressed inter-enterprise business processes while commercial workflow products and workflow standards have been confined to system-centered approaches and to the enhancement of workflow systems with Internet technologies.

Like in the previous chapter, here the convergence of different trends of development has dictated the evolution of workflow management. The development of the World Wide Web and its associated technologies together with system-centered workflow research has led to a new generation of Web-enabled workflow systems referred to as Internet-mediated workflow. Internet-mediated workflow concerns basically the same internal business processes as system-centered workflow, but now with improved accessibility and ubiquity provided by Web-based approaches. This trend, together with business networking, led to inter-enterprise workflow, i.e., the coordination of business processes across several business partners and supported by the Internet and Web technologies. The fact that workflow management has so much to benefit from the Internet and its associated technologies is due to an intrinsic characteristic of workflow management systems: the fact that these systems pose strong requirements on the underlying integration infrastructure.

Going back to figure 3.2 on page 65, it can be seen that workflow management is built on three main assumptions:

1. the assumption that business processes can be analyzed and modeled,

	WORKFLOW RESEARCH PROJECTS	COMMERCIAL WORKFLOW PRODUCTS	WORKFLOW STANDARDS
SYSTEM-CENTERED WORKFLOW	Transactional capabilities; distributed object-oriented architectures	Database-centered and client/server architectures	WfMC and OMG interfaces and standards
INTERNET-MEDIATED WORKFLOW	Task delivery through the Web and e-mail; Web-based interface to workflow systems		Binding of WfMC standards; SWAP
INTER-ENTERPRISE WORKFLOW	Coordination of inter-enterprise processes; support for electronic contracts	?	?

Figure 3.35: Lack of products and standards in inter-enterprise workflow

2. the assumption that it is possible to control the repeated execution of those business processes, and

3. the assumption that it is possible to invoke, interact or exchange information with all resources involved in those business processes.

These three assumptions have different implications but, while the first two assumptions can be addressed by a single entity (a team of process designers, for example), the third assumption requires all relevant resources to be able to exchange and understand information from each other. If these resources cannot exchange information and understand each other then the workflow management system, whose purpose is to manage the flow of work from resource to resource, will be unable to execute business processes. A workflow management system thus requires resources to be integrated prior to process enactment. This challenge is aggravated when business processes become more comprehensive and encompass more resources. Business processes that cross several departments require the integration of heterogeneous resources on an enterprise-wide scale. Inter-enterprise business processes require the integration of internal resources with external resources that belong to other enterprises.

Whatever the scope of business processes that a workflow management system supports, process enactment requires an underlying integration infrastructure that facilitates information exchange between all relevant resources, and between these resources and the workflow management system. This requirement affects every workflow system. Staffware relies on an infrastructure based on remote procedure calls, while OfficeWorks relies on a set of communication services including fax and e-mail. Research prototypes employ a variety of application integration technologies such as object-oriented databases, CORBA, RMI, HTTP, CGI scripts, and Java servlets and applets. The fact is that every workflow management system has its own integration infrastructure, either by relying on another system or by devising its own integration architecture. For example, TriGSflow relies on a object-oriented database system, SWORDIES relies on the EvE execution system, WIDE and WASA rely on relational database systems, MENTOR relies on a TP monitor, EXOTICA relies on IBM MQSeries. On the other hand, APRICOTS has developed its own Communication-System module, METEOR has devised its own CORBA-based infrastructure, PRODNET has devised its cooperation layer (PCL), VEGA has developed its COAST communication platform, ACE-Flow and WISE have

built their own centralized Web catalogs, WHALES relies on its Web-based project environment, COSMOS has implemented its own Contract API, and CrossFlow has designed its own CRAFT integration infrastructure.

Given that workflow management systems require an integration infrastructure, it is thus not surprising that the Internet and its associated technologies have had such a strong impact on the capabilities of those systems. On one hand, these technologies will allow workflow management systems and their functionality to be distributed across a network. On the other hand, they will allow resources to perform work anytime, anywhere. In fact, without such a ubiquitous and easily accessible network it would be extremely difficult and expensive to come up with a common infrastructure which enterprises could use to integrate resources across their borders. The Internet and its associated technologies excel in providing access to and in integrating remote applications, and they can and should be used as the basis for the underlying integration infrastructure for inter-enterprise workflow management.

In workflow management there is always a duality between the workflow system and its underlying infrastructure. In many cases, this duality is not obvious because both components have been devised together, and it is difficult to say where the workflow system ends and the integration infrastructure begins. For example, in a workflow system architecture such as METEOR it is difficult to separate the integration infrastructure, which is based on CORBA, from the workflow system, because the workflow system itself is based on CORBA. This tight coupling is effectively a limitation of workflow management systems since, in general, a given workflow engine cannot be used with a different integration infrastructure. As a result, many workflow management systems, however sophisticated and ingenious they may be, find limited applicability, not because they lack appropriate functionality, but because they are bound to specific integration infrastructures, from which they cannot be detached.

From the survey of commercial workflow products, two approaches concerning the development of workflow management systems have been recognized. On one hand, there are those which start from business process modeling and enactment, and subsequently develop the necessary infrastructure in order to interact with resources (top-down approach). On the other hand, there are those which focus first on the infrastructure, and only afterwards consider functionality for modeling and enacting the business processes they support (bottom-up approach). The first approach is the most frequent in commercial workflow products, and it completely dominates research on workflow management systems. Since the main focus of workflow management has been put on process modeling and execution, these features have often received more attention than the underlying integration infrastructure. But because every workflow management system requires an integration infrastructure, such an infrastructure has been devised or chosen for each workflow system, irrespective of the fact that it would eventually become a limiting factor regarding the applicability of the system.

This work realizes the duality between the workflow system and its underlying integration infrastructure, and adopts a separate development approach for these two components. In essence, it is believed that workflow management concerns mainly process modeling and execution, and that the integration infrastructure concerns resource integration. Standardization efforts in workflow management have addressed only process modeling and execution. Regarding the integration infrastructure, the Workflow Reference Model states that this area requires further specification work [WfMC, 1995]. In fact, the integration of enterprise resources brings a whole new set of issues and challenges. These issues require not only architectures, but also methodologies that support the full cycle of development and implementation of those architectures, which facilitate information exchange and coordination across a wide range of heterogeneous resources. Incidentally, this area has already been the focus

of extensive research. Although most approaches in workflow management have not always been aware of the duality between processes modeling/execution and integration infrastructures, a related field of research has always realized the need for both components and the fact that these components complement each other. This research field is called Enterprise Modeling and Integration or, in short, Enterprise Integration, and it is the focus of the next chapter.

Chapter 4

Perspectives from Enterprise Integration

The analysis and definition of business processes produces explicit process models that describe the relationship among a set of tasks. This analysis promotes an increased awareness of how the work is structured and opens opportunities for improvement. It is possible that, when introducing the use of a workflow management system, process definitions will exhibit bottlenecks, limitations, or other issues suggesting the need for improvement. For example, some steps may be found unnecessary, or a single step may be identified as the major cause for delay in running a certain procedure. A workflow system may contribute to define and employ new business procedures or work structures. Additionally, it provides a framework for the automation of those business processes. It is thus not surprising that workflow management has often been discussed together with business process reengineering, computer integrated manufacturing, and enterprise modeling and integration approaches [Totland and Conradi, 1995].

The framework of workflow management includes process execution capabilities that control and track the progress of process instances during run-time. This means that workflow automation can only succeed if the workflow management system is able to exchange commands, data and events with resources assigned to activities. Because every resource relates to a business process in one form or another, and because business processes may be improved, reengineered or completely changed, a workflow management system can only enjoy a prolonged utilization if it is able to interact with all resources, current and future ones too. If the system becomes unable to account for the automation of certain activities then it will be incapable of coordinating the respective business processes.

Workflow management thus imposes severe infrastructural requirements because it can only be employed if resources are already integrated to a certain degree, or if that integration is feasible. The ideal scenario would be to already have an enterprise integration infrastructure that allows resources to interact with each other. On top of that infrastructure, a workflow management system would orchestrate the action of resources, by requiring certain tasks to be performed and by routing information from one to another until business processes are complete. This is a key concept that will be fundamental in the course of this work.

However, workflow management alone, which had its origin in the routing of documents inside an office, does not address these infrastructural challenges. Instead, the development of enterprise integration architectures and infrastructures has been the purpose of Enterprise Integration (EI) [Vernadat, 1996]. This discipline had its origin in Computer Integrated Manufacturing (CIM) [Ranky, 1986] during the 1980s, and throughout the years has evolved from devising manufacturing systems to developing methodologies that address the enterprise and its business processes as a whole. Enterprise Integration may be summarized as follows (adapted from [Vernadat, 1996]):

> Enterprise Integration (EI) is concerned with facilitating information, control, and material flows across organizational boundaries by connecting all the necessary functions and heterogeneous resources in order to improve communication, cooperation, and coordination within this enterprise so that the enterprise behaves as an integrated whole, therefore enhancing its overall productivity, flexibility, and capacity for management of change.

Enterprise Integration can be classified as comprising three different levels: physical integration, application integration, and business integration. Physical integration refers mainly to systems interconnection and data exchange by means of computer networks and communication protocols, such as the ones provided by the Internet and its associated technologies. Application integration concerns the interoperability of applications on heterogeneous platforms as well as access to shared data. Business integration pertains to business process design and coordination. These levels are layered on top of each other from lower to higher integration level, as illustrated in figure 4.1.

Much of today's e-business literature pays no attention to the contributions of Enterprise Integration, and does not take into account systematic approaches that have deserved considerable thought in recent past. Emerging e-business integration approaches are often based on technologies such as XML but get little further than application integration. Meanwhile, Enterprise Integration realizes the benefits of emerging technologies and adopts them to further enhance existing integration architectures [Zelm, 1999]. On the other hand, workflow management is sometimes applied without realizing its infrastructural requirements, resulting in restricted solutions because they are premature attempts towards business integration[58]. Enterprise Integration provides a conceptual framework showing the leap between application integration and business integration - the step that many e-business approaches still lack - and studies architectures where both kinds of integration play an important role.

During the 1980s and the 1990s, Enterprise Integration has been characterized by the development of several integration architectures. These architectures provide tools and methodologies to break down systems into functions, to understand the relationships between those functions and to identify generic building blocks that can be used to build any of those functions. More than aiming at a particular manufacturing system or enterprise infrastructure, an integration architecture addresses the challenge of enterprise integration from a higher level of abstraction that allows methodical approaches to take place, regardless of the details of the particular system. Each enterprise integration architecture is therefore a *reference architecture,* which defines a methodology to capture enterprise functions into models, and to devise particular system architectures that implements those models.

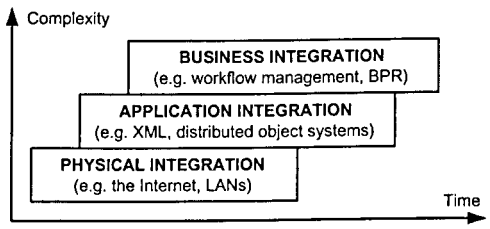


Figure 4.1: Levels of enterprise integration[57]

[57] adapted from [Vernadat, 1996]

[58] This situation is analog to that of "islands of automation" in CIM.

Different reference architectures propose different approaches towards enterprise integration, so having insights from distinct architectures is helpful in order to capture the essence of enterprise integration. In the following sections, the discussion will be restricted to CIMOSA, GERAM and EMEIS, which are thought to be the most relevant to this work. Each reference architecture constitutes an extensive subject per se, stimulating reflection on a wide range of conceptual and technical issues. Following an overview of those three reference architectures, section 4.4 discusses how enterprise integration concepts relate to workflow management and business networking.

4.1 CIMOSA

The Open System Architecture for CIM (CIMOSA) [AMICE, 1993] was the result of a major research project on Enterprise Integration focusing on the manufacturing industry. The project was known as AMICE[59] and had 28 participating organizations from 9 European countries. The goal of AMICE was to develop an Open System Architecture for integrating enterprise internal operations, while supporting the migration of existing (and forthcoming) systems towards that Open System Architecture. The AMICE members realized that the Open System Architecture would have to meet the following requirements:

- to provide an architecture that describes the real world of the manufacturing enterprise;
- to allow users to directly control and maintain enterprise systems without requiring IT expertise;
- to provide appropriate concepts to structure an enterprise so as to minimize the impact of change;
- to provide selective models of an enterprise which emphasize all relevant aspects;
- to distinguish between the tasks of updating or modifying enterprise systems from the tasks of operating those systems;
- to provide an information technology infrastructure enabling the portability of computer applications and system-wide information exchange, while supporting hardware and software from multiple vendors.

The last three points have had a crucial role in devising CIMOSA and correspond directly to the three concepts the characterize CIMOSA: a modeling framework, the distinction of operation and engineering environments, and the concept of an integrating infrastructure. In particular, the last point seems almost premonitory in view of the middleware infrastructures available today, which are close to fulfill that goal.

4.1.1 CIMOSA Modeling Framework

While modeling the enterprise there are so many aspects and viewpoints to be considered that they must be arranged into a multi-dimensional framework. The CIMOSA modeling approach is based on three dimensions. At first, the dimension of *view* divides models into submodels that pertain to different aspects of the enterprise. Within the view dimension, models are grouped according to four

[59] European Computer Integrated Manufacturing Architecture (reverse acronym)

distinct views. The *function view* allows observation of the enterprise functionality for operation, planning, control and monitoring. The *information view* allows observation of the structure of business information used by those functions. The *resource view* allows observation of the enterprise assets needed to perform the enterprise business processes. The *organization view* allows observation of decision making responsibilities.

The second dimension is that of *modeling levels*, comprising the requirements definition, design specification and implementation description. Requirements definition models identify business requirements; it defines what has to be done to meet the enterprise objectives, regardless of the technology to be employed. Design specification models describe, in an implementation independent format, the technology required to perform the identified processes. Implementation description models indicate the actual selection of vendor products providing the information technology and manufacturing technology components required for the execution of those processes.

The third and last dimension is that of *genericity* of models. Starting from right to left in figure 4.2, *particular models* are concerned with a particular enterprise and these models can be used directly to derive requirements, design and implementation models that address specific business or manufacturing challenges for that particular enterprise. *Partial models* are more generic, incomplete skeletons of models that can be applied to more than one industrial sector, company organization or manufacturing strategy. *Generic building blocks* constitute a further level of abstraction and provide a set of basic constructs that can be reused in building any enterprise model. CIMOSA provides the generic building blocks summarized in figure 4.3.

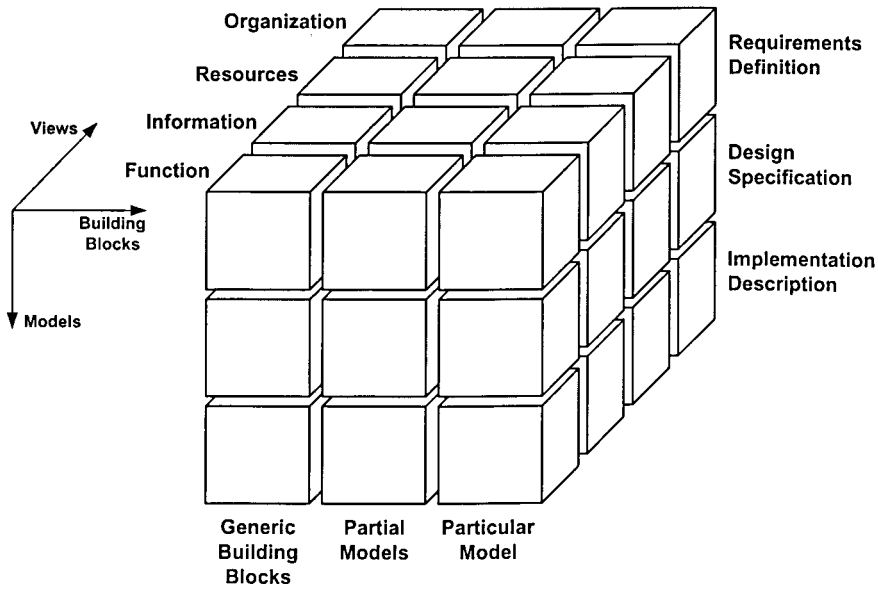


Figure 4.2: CIMOSA Modeling Framework[60]

[60][Vernadat, 1996]

Organization	Resources	Information	Function	
Domains Events	Enterprise objects Information Elements	Capabilities	Responsability Authority	**Requirements Definition**
Specified Functional Operations	External Schemata Conceptual Schema	Specified Capabiblities Specified Resources	Organization Units Organization Cells	**Design Specification**
Implemented Functional Operations	Implemented schemata Internal Schema	Implemented Capabilties and Resources	Implemented Organization Units and Cells	**Implementation Description**

Figure 4.3: CIMOSA Generic Building Blocks[61]

4.1.2 CIMOSA Environments

While trying to render the complexity of enterprise operations more manageable, CIMOSA defines two mutually independent but related environments: the *enterprise operational environment* and the *enterprise engineering environment.* The enterprise operational environment is where products are developed, produced, marketed and sold, orders are processed and bills are paid. The enterprise engineering environment is where particular business processes are modeled, simulated, evaluated, and released for operation. In other words, these environments decouple the engineering of enterprise systems (engineering environment) from the daily enterprise operations (operation environment).

Each of these environments has an intrinsic life cycle. The operation environment takes care of the product life cycle, the set of different phases of the product's lifetime. The engineering environment handles the system life cycle concerning the definition, implementation and maintenance of procedures and system components that support the tasks of the product life cycle. The operational phase of the system life cycle is the execution of the product life cycle, as suggested in figure 4.4. By assigning enterprise tasks to one life cycle or another, it is possible to structure the operations of the enterprise so as to handle change more easily.

4.1.3 CIMOSA Integrating Infrastructure

The CIMOSA Integrating Infrastructure provides the mechanisms to integrate heterogeneous resources. More importantly, it also provides the means to control the life cycle processes via the execution of CIMOSA-compliant enterprise models. The CIMOSA Integrating Infrastructure comprises five interacting sets of services: the Business Services, the Information Services, the Presentation Services, the System Management Services, and a set of Common Services. These services are implemented over IT Supporting Services, as shown in figure 4.5.

The Business Services assume that enterprise operations are event-driven and focus on their coordination, sequencing, and synchronization. These services also aid in managing enterprise resources and allow human intervention on exceptional events. Basically, Business Services provide the functions required to execute enterprise models. A Business Process Control Service (BC) interprets procedural rule sets and dispatches the execution of enterprise activities. The Activity Control Service (AC) interacts with resources and provides them with functions necessary to carry out their tasks.

[61] [Vernadat, 1996]

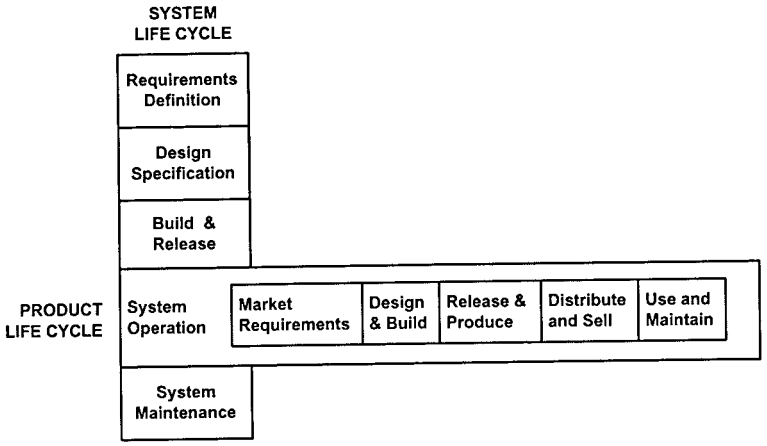


Figure 4.4: CIMOSA life cycles[62]

A Resource Management Service (RM) reserves and dynamically schedules the resources assigned to enterprise activities.

The three Business Services interact in the following way. An event triggers the BC service and a business process occurrence (i.e. instance) is created. The BC service requests the RM service to schedule that process occurrence. When the first enterprise activity reaches its start time, the BC service will ask the AC service to execute it. The AC service will then ask the RM service for functions that are necessary for that enterprise activity. Afterwards, the AC service will request the assigned resource (human, machine or application) through Presentation Services to execute the enterprise activity. The Presentation Services control the execution of the enterprise activity and reports its results back to the AC service, which will forward it to the BC service. The BC service will then request the AC service to execute the next activity. After the last activity is performed, the BC service issues an end event for the process occurrence.

The Presentation Services provide functions required to control the execution of enterprise activities by heterogeneous resources. There are three Presentation Services - the Machine Dialogue Service (MD), the Application Dialogue Service (AD), and the Human Dialogue Service (HD) - that

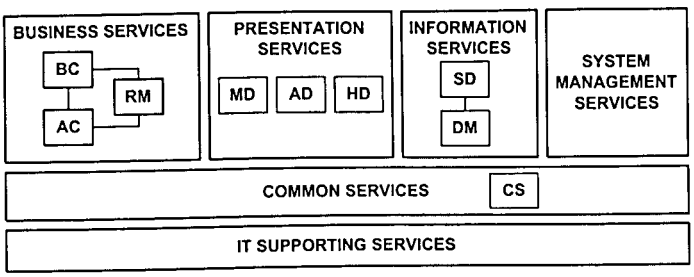


Figure 4.5: The CIMOSA Integrating Infrastructure[63]

[62][AMICE, 1993]
[63]adapted from [AMICE, 1993]

are able to interact with the three corresponding types of resources. For example, the MD service should be able to exchange instruction scripts with programmable controllers.

The purpose of Information Services is to support system-wide access to enterprise information, while providing transparent access to heterogeneous data storage means. Information Services support the definition, manipulation and change of enterprise information. These services also aim at maintaining consistency and integrity of enterprise information, and enforcing information access rights. A System Wide Data Service (SD) provides a unified data access to its clients through the definition and use of SQL views. The SQL views are based on a global schema which may refer to data distributed over several system nodes. The data requests are split, if necessary, into a set of local requests according to the distribution of data. The Data Management Service (DM) is responsible for presenting any storage system in a transparent way to the SD service.

The Common Services ensure interoperability between services of the infrastructure by providing message exchange and allowing certain levels of transparency. The Communication Service (CS) is a message exchange facility: it handles message queues, allows synchronous and asynchronous communication, and ensures message transfer, recovery and re-delivery if necessary. Naming and registration services for application components are also included. Common Services provide location transparency (components may be local or remote), access transparency (the same access mechanism for local or remote components), migration transparency (nothing changes if components are moved), replication transparent (clients are unaware of which server is providing the service), and concurrency transparency (concurrency in components is invisible to other components)[64].

Finally, System Management Services provide generic facilities to set up, maintain, and monitor the components of the integration infrastructure. System Management Services allow new components to be installed and configured for run-time operations. CIMOSA does not specify these services in detail, referring only to the fact that they should support all management functions applicable to configuration, performance, security and fault management.

In conclusion, a CIMOSA Integrating Infrastructure should be devised with the following features in mind: distribution, openness, portability of applications, connectivity, compliance with standards, reliability, and performance.

4.2 GERAM

Research on enterprise integration has produced three major architectures - CIMOSA, GRAI-GIM [Doumeingts et al., 1987] and PERA [Williams et al., 1996] - each one proposing its own framework and methodology for the development and implementation of enterprise integration programs. It can be shown that these architectures have several features in common [Vernadat, 1996]. For example, all three architectures address the requirements phase, design phase and implementation phase (these are called modeling levels in CIMOSA and abstraction levels in GRAI-GIM), which are a subset of the PERA enterprise life cycle. On the other hand, each architecture has something unique to offer, for example, the integrating infrastructure in CIMOSA, the concept of decision centers in GRAI-GIM, and the human and organizational architecture in PERA.

Realizing the differences and similarities between these architectures, the IFAC/IFIP Task Force on Architectures for Enterprise Integration has developed a generalized architecture, the Generalized Enterprise Reference Architecture and Methodology (GERAM) [IFIP-IFAC, 1999]. The purpose of GERAM is to become a reference for the whole enterprise integration community, defining the

[64] All these features are commonplace in modern distributed object systems (e.g., CORBA).

terminology, providing a generalized methodology, and giving a unifying perspective of enterprise integration architectures and related products. GERAM captures the essence of existing architectures and presents a framework for comparing and evaluating enterprise integration approaches.

There are three main uses of GERAM. GERAM, as with the other architectures it is based upon, can be used as the basis for developing enterprise integration programs, either by employees of a particular enterprise or by consultants working for an enterprise. But because GERAM is a generalized architecture and methodology, it can be used also to develop new and improved methods or tools to assist those integration programs. Finally, GERAM can be used by researchers, academics, and others developing new modeling theories and ontologies, as the theoretical underpinning for integration programs or for new methods and tools that have resulted from applying enterprise integration approaches, based on GERAM or other integration architectures. This third use is the most relevant to this work, as GERAM allows to identify and characterize the features of a particular integration approach.

4.2.1 GERAM Framework components

GERAM provides a description of all the elements and their relationships within an enterprise integration architecture. These elements, referred to as framework components, comprise the concepts, tools, formalisms, and models necessary for any integration program. Figure 4.6 illustrates the relationships between the following set of framework components:

1. The GERA (Generalized Enterprise Reference Architecture) defines generic concepts for enterprise integration. These concepts are divided into human-oriented concepts (an influence from PERA), process-oriented concepts (from CIMOSA), and technology-oriented concepts (corresponding to the technically-oriented approach of GRAI-GIM).

2. Every enterprise architecture employs a particular methodology, i.e., a structured approach with detailed instructions for each integration activity. This methodology is represented in GERAM as the Enterprise Engineering Methodology (EEM).

3. The methodologies for enterprise integration rely on models to describe the structure and behavior of the enterprise, both as-is and to-be. For this purpose, GERAM includes Enterprise Modeling Languages (EMLs) that provide constructs for enterprise modeling. Modeling constructs describe and model human roles, operational processes and their functions, information elements, and office and production technologies.

4. Generic Enterprise Modeling Concepts (GEMCs) define the semantics of enterprise modeling languages and its constructs. GEMCs formalize generic concepts of enterprise modeling and describe the relationship among those concepts so that they can be employed correctly and unambiguously in enterprise models.

5. Partial Enterprise Models (PEMs) are reusable models that capture typical characteristics common to many enterprises. They are repetitive or systematic results from enterprise integration programs and can be reused instead of modeling the enterprise from scratch. Partial models may extend to all fields of the generalized enterprise architecture (human roles, operational processes or technology components).

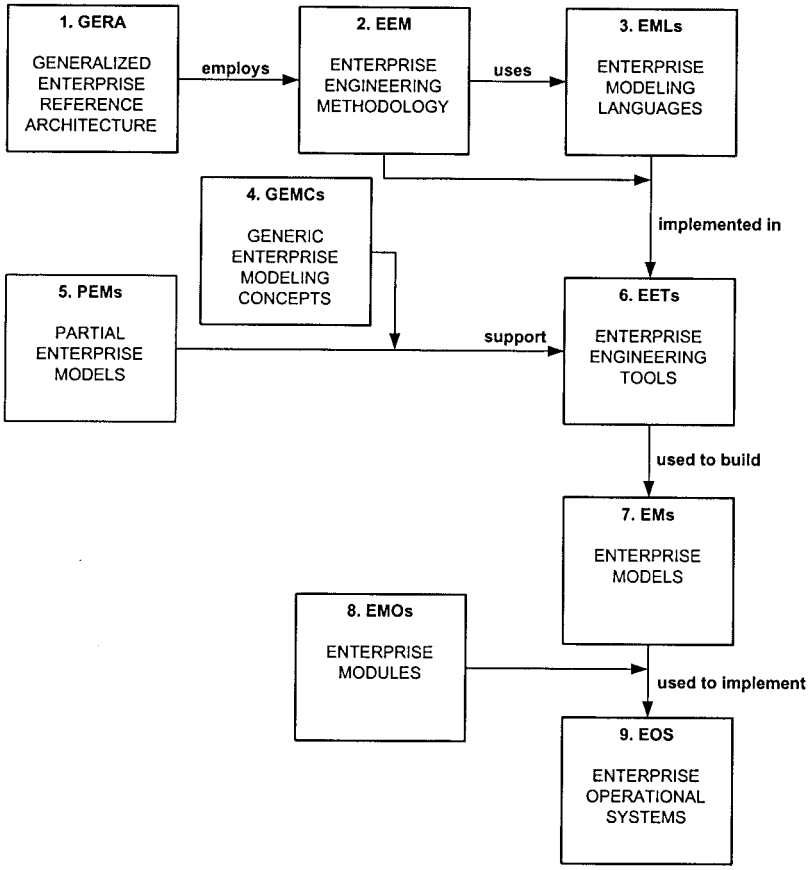


Figure 4.6: GERAM framework components[65]

6. Enterprise Engineering Tools (EETs) are support tools for enterprise integration. These tools implement the integration methodology and its modeling languages. EETs work according to the semantics of generic modeling concepts and allow the construction of models from partial models. EETs support the analysis, design and use of enterprise models.

7. Enterprise Models (EMs) use modeling languages to describe a particular enterprise. These models may consist of several views describing distinct aspects of the enterprise (like the CIMOSA views). They may also include models for different purposes such as models for analysis, models for design or models for implementation.

8. Enterprise Modules (EMOs) are products or resources supporting the implementation of the enterprise. They can be human resources, manufacturing resources, business equipment or information technology infrastructure. According to GERAM, these modules should support the operational use of enterprise models.

9. Enterprise Operational Systems (EOS) are particular systems supporting the operation of a particular enterprise. The implementation of operational systems is based on enterprise models which specify those systems and identify the required enterprise modules.

[65] [IFIP-IFAC, 1999]

GERAM framework components involve a significant amount of technology through Enterprise Engineering Tools (EETs), Enterprise Modules (EMOs) and Enterprise Operational Systems (EOS). Technology is present in the engineering environment where integration programs are developed, and in the operational environment where production, management and control take place. In both these environments, information technology provides the means for communication, information processing and information sharing. Also in both of these environments, technology concerns concepts that relate to Enterprise Models (EMs) - technology is used either to build enterprise models and to implement them. GERAM thus requires information technology to:

- ensure model portability and interoperability by providing an integration infrastructure across heterogeneous enterprise environments;
- enable model-driven operational support by providing real-time access to the enterprise environments.

4.2.2 GERAM Entity life cycles

GERAM considers the enterprise as an entity and allows this entity to contain or manage other entities. This way, an entity can be any kind of endeavor such as a product, a project, a methodology or an enterprise. Furthermore, GERAM realizes that the life cycles described by CIMOSA and PERA refer to enterprise entities. For example, the CIMOSA system life cycle and product life cycle depicted in figure 4.4 are the same life cycle applied to two distinct entities. However, in order to cover a generic life cycle that can be applied to any entity, GERAM adopts a life cycle that closely resembles PERA's, which is more complete. The adopted life cycle is shown in figure 4.7.

The identification phase comprises a set of activities that identify the scope and contents of the particular entity under consideration. Those activities include the identification of the need for this entity. The concept phase comprises another set of activities that develop the concepts for the entity. This corresponds to PERA's concept phase where the mission, vision and values are defined. The requirements phase describes the operational requirements for the enterprise entity. The design phase elaborates the specification of the entity and of the required components and resources that satisfy the operational requirements. The implementation phase consists in commissioning, developing, purchasing and configuring all components, possibly hiring and training personnel or changing the human organization. The operation phase represents the control, monitoring and evaluation of business processes and resource operations so as to fulfill the entity's mission. Finally, the decommission phase deals with re-design, reengineering, disassembling, recycling and possible disposal of all or part of the entity.

Also shown in figure 4.7 is the fact that the operation phase of one entity may consist in, or have a relationship to, the life cycle of another entity. This is similar to the relationship between system life cycle and product life cycle in CIMOSA, but GERAM allows this relationship to be extended without restrictions to other entities and to other life cycle phases as well. It is then said that the operation of one entity supports the life cycle activities of another. For example, entity A may be an engineering entity whose operation is the design and implementation of an entity B that could be a manufacturing plant. Other relationships between life cycle activities of two entities can be defined as well.

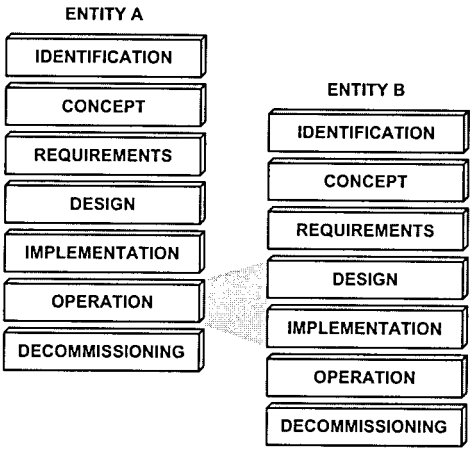


Figure 4.7: GERAM life cycle phases[66]

4.2.3 GERAM Enterprise entities

Because an entity is any kind of enterprise endeavor, there may be several ways to group and categorize them. GERAM defines a generic and recursive set of five enterprise entity types. Four of these enterprise entity types are:

- the Strategic Enterprise Management Entity (Type 1), which identifies and defines the need for starting an enterprise engineering or integration effort;
- the Enterprise Engineering/Integration Entity (Type 2), which performs the integration program defined by an enterprise entity of Type 1;
- the Enterprise Entity (Type 3), which is a set of business processes and resources that result from the operation of entity type 2 and concern the manufacturing of products and rendering of services;
- the Product Entity (Type 4), which is the outcome of the operation of entity type 3 and represents the products and services of the enterprise.

These four entities can be understood in the following way. A product or service is something that must be designed, manufactured and operated (the situation for services is similar), hence the life cycle of entity type 4. These products are manufactured by an enterprise that comprises its office and manufacturing plants; these entities must be engineered as well, hence the entity type 3. Now, GERAM considers an enterprise (its structure and facilities) to be the result of one or more enterprise engineering/integration efforts, something that has to be specified and implemented, thus something that possesses its own life cycle, entity type 2. The first entity - Type 1 - is simply the need or opportunity that led to the enterprise engineering effort, according to strategic business goals.

The fifth entity type represents the methodology that guides the enterprise engineering/integration effort and the activities undertaken within its framework (entity type 2). The methodology will produce manufacturing, information and human architectures that will shape the structure and behavior of the enterprise (entity type 3). The fifth entity type is therefore:

[66][IFIP-IFAC, 1999]

- the Methodology Entity (Type 5), which represents the approach to be employed by any entity type during its operation, which may lead to the creation of another entity type.

Figure 4.8 illustrates the five entity types, their life cycles and the relationship between them.

4.3 EMEIS

The starting point for every enterprise integration architecture, as previous sections have shown, is to describe the enterprise and its business processes by means of models. CIMOSA uses models to conduct the design and implementation of enterprise systems. GRAI-GIM uses models to convey user-oriented requirements and to develop technically-oriented specifications of enterprise systems. PERA uses models to derive an enterprise integration master plan that includes three different system architectures. And GERAM uses models to support the life cycle of enterprise entities. The ability to build models of the enterprise and of its business processes is a cornerstone of Enterprise Integration.

In enterprise integration architectures such as CIMOSA, GRAI-GIM and PERA, the purpose and usage of enterprise models differs slightly. In general, however, enterprise models are used for guiding the design and implementation of enterprise systems. GERAM identifies Enterprise Models

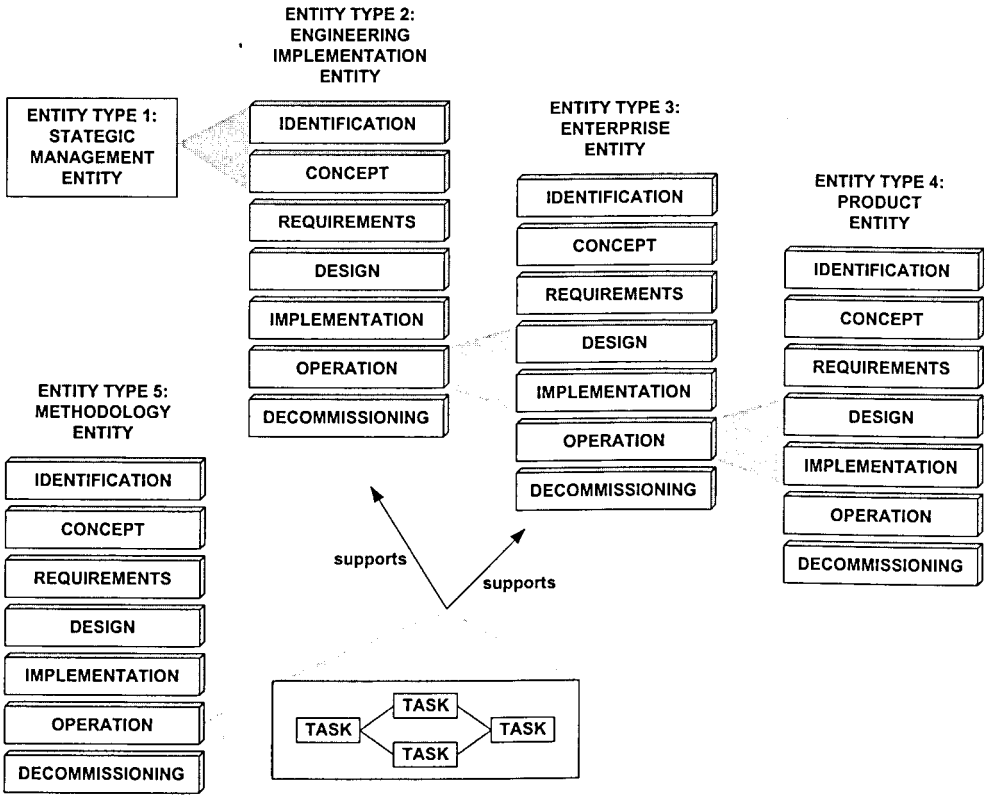


Figure 4.8: Relationships between life cycles of GERAM entity types[67]

[67] [IFIP-IFAC, 1999]

(EMs) and Enterprise Modules (EMOs) as the two framework components needed for implementing Enterprise Operational Systems (EOS), as shown in figure 4.6. But GERAM extends the life cycle beyond implementation and into operation, and defines a Methodology Entity (entity type 5) as an entity that supports the complete life cycles of enterprise entities, as depicted in figure 4.8. As a consequence, GERAM does not confine enterprise models to design and implementation, but realizes that there is role for enterprise models during the operation phase as well.

In fact, and as presented earlier in figure 4.5 on page 144, CIMOSA had already anticipated this possibility by including in its Integrating Infrastructure a set of Business Services (especially BC) that provide the functions required to execute enterprise models. These models are not specification models or implementation models; they are *executable models*, i.e., computer-processible enterprise models used to control operational activities. Executable models are an abstraction of how the enterprise system operates. This abstraction is used not only to describe but also to drive the operation of the enterprise system. The concept of executable models takes the purpose and usefulness of Enterprise Modeling to the farthest extent possible.

The usage of executable models closes the gap between the enterprise engineering and enterprise operation environments. Without executable models, enterprise integration is a one-way endeavor: enterprise engineering identifies requirements, designs the system, and implements it, while enterprise operation operates it. Whenever the system has to be changed, whether improved, redesigned or completely reengineered, the whole life cycle repeats itself back from beginning by identifying the new requirements. On the other hand, employing executable models for driving the enterprise system operation establishes an iterative relationship between engineering and operation by allowing models to be changed while leaving the run-time infrastructure for those models fundamentally intact. Whenever the enterprise system must be changed, enterprise engineering redesigns the appropriate models so as to come up with new executable models without having to reengineering the system from the beginning. Figure 4.9 illustrates this advantage.

Executable models are the natural extension of enterprise models to the operational environment. But executable models differ from other models in that they chain enterprise activities that are assigned to operational, real-world resources. Therefore, in order to be effective, executable models require an appropriate run-time infrastructure. This run-time infrastructure must provide the appropriate services for scheduling business processes, initiating and terminating enterprise activities, handling events, and interacting with resources. Moreover, the infrastructure must be extended to the whole enterprise entity and all of its resources in order to allow the timely coordination, sequencing and synchronization of that entity's business processes. Consequently, executable models come hand-in-hand with infrastructures that provide the run-time environment for their execution.

CIMOSA had already realized these infrastructural requirements by proposing an integration infrastructure with a set of information technology services supporting the execution of enterprise models. GERAM requires information technology to provide an integration infrastructure across heterogeneous enterprise environments in order to enable model-driven operational support. Clearly, the development of information technology infrastructures that support the execution of enterprise models is just as important as the development of those executable models, and they cannot be dissociated from each other. Executable models cannot fulfill their purpose if they lack the infrastructural support for their execution, and integrating infrastructures alone are not enough to achieve business process integration.

Enterprise integration thus requires infrastructures that provide two main sets of services: Model Development Services and Model Execution Services. Indeed, standardization efforts within the field

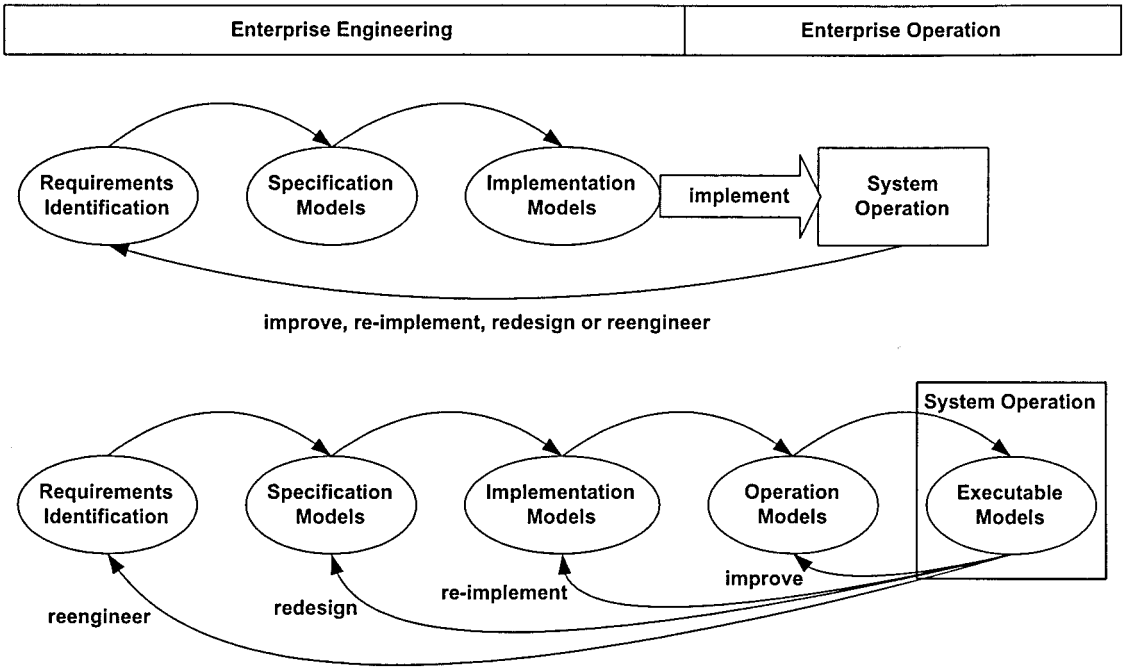


Figure 4.9: Enterprise integration with and without executable models

of Enterprise Integration have led to the framework of Enterprise Model Execution and Integration Services (EMEIS) [CEN, 1998] that includes the following sets of services:

- Model Development Services (MDS) - these services support the cooperative development and analysis of models, partial models and executable particular models of the enterprise entity;
- Model Execution Services (MXS) - these services support the operational use of models, the enactment, control and monitoring of the enterprise entity's operations, according to executable models.

In addition to these elements, EMEIS identifies two other sets of services: Shared Services that apply to both the engineering and the operational environments, and Base IT Services which are fundamental information technology services that serve as basis for developing the other services.

4.3.1 Model Development Services (MDS)

The framework of EMEIS identifies four phases for model development, as shown in figure 4.10:

- Phase 1 concerns the overall enterprise architecture. It identifies enterprise domains and the relationship between them. The relationships between enterprise domains may involve the exchange of events, information, materials, or products.
- Phase 2 decomposes enterprise domains into their constituent business processes. These business processes can be obtained from a bottom-up study and assembly of the domain activities.

Alternatively, business processes may be decomposed according to a top-down modeling approach into their constituent domain activities. In either case, the result is a set of activity chains that describes the business processes for the relevant domains.

- Phase 3 focuses on the definition of each domain activity, and on what their inputs are and outputs are. EMEIS considers activities as having functions, controls and resource inputs and outputs. These can be compared to the PERA generic task module [Williams et al., 1996]: functions correspond the transformation process, controls are enabling parameters, and resources are the human, machine or information capabilities used during the transformation process. EMEIS also relates each activity to a responsible organization unit.
- Phase 4 defines the enterprise objects, resource objects and organizational units that are functions, controls or resources of business process activities. The purpose of phase 4 is to identify and describe these objects as non-redundant and reusable structures. This will lead to building blocks and partial models that can be used to model additional views of the enterprise using specific enterprise modeling constructs [CEN, 1996].

According to EMEIS, the Model Development Services (MDS) should support these phases and the development of their corresponding models by providing assistance in model creation, model assessment, model management and in maintaining a model repository.

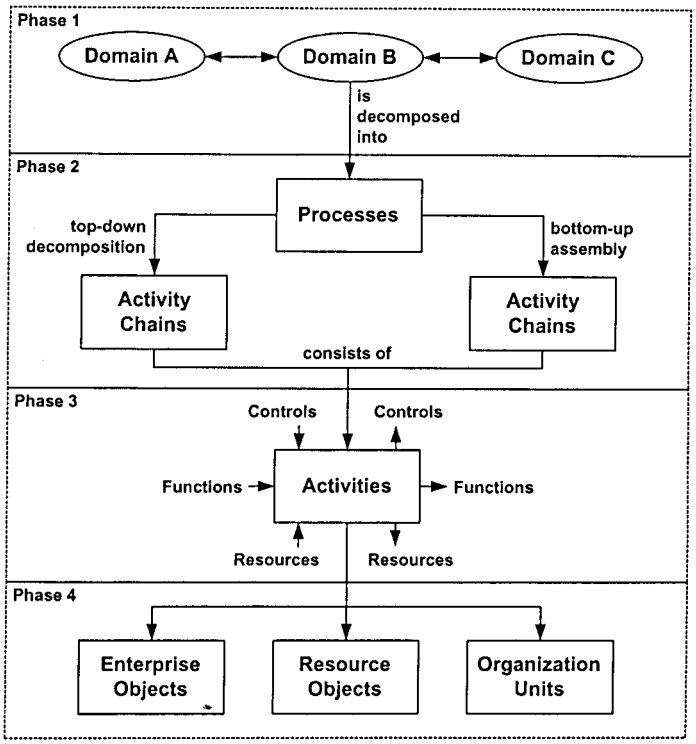


Figure 4.10: EMEIS phases of model development[68]

[68][CEN, 1998]

Model creation functionality covers all the necessary features for creating construct types, partial models, and particular models, in addition to general features such as browsing, navigating, visualizing and selecting model components and model views. The creation of construct types means defining and specializing basic building blocks and their associated semantics. These building blocks will then be used to compose model components, which in turn can be combined into partial models. Particular models are built from combinations of partial models and additional model components, in order to describe the relevant enterprise objects or resource views.

Model assessment allows the analysis, simulation and evaluation of particular models. The analysis of models addresses such characteristics as structure, consistency and completeness. Dynamic test facilities such as Petri nets allow to check for additional properties such as liveness and reachability. Simulation is used to anticipate the behavior of models and can be used as a design support tool to verify the desired behavior, or as decision support tool to test alternative scenarios. EMEIS allows simulated models and enacted models (executable models under execution) to run simultaneously, although in different contexts or work environments. The simulation of models, as well as their enactment, is handled by Model Execution Services (MXS). Model Development Services (MDS) are used in preparing and designating the execution context, in checking for model consistency and completeness prior to simulation runs, and in generating artificial events both to trigger and in the course of a simulation run. After being simulated, models can be evaluated according to criteria such as performance, processing times, associated costs, or storage requirements.

Model management characterizes model components, partial models and particular models in order to support model browsing, selection, retrieval and deletion. EMEIS proposes these models to be placed under a class hierarchy with external interfaces which determine under what conditions each model can be used, and the relationships between those models. Model management supports the development of models through version control, records which models have been released for simulation or enactment, and provides information about models to MDS and MXS services.

Model management works in articulation with a model repository that maintains named instances of models and representations of their internal structure. The model repository also registers functional entities (humans, machines, applications) and the appropriate form of presenting models to those entities. The model repository allows models to be organized and searched within the class hierarchy.

4.3.2 Model Execution Services (MXS)

The purpose of the Model Execution Services (MXS) is to enable the timely coordination, sequencing and synchronization - the enactment - of enterprise operations described by executable models. This enactment is carried out by MXS in response to monitored events and according to sequencing rules. Rules specify the behavior of business processes and enterprise activities that concern the enterprise operations. EMEIS adopts the following definition of business process [CEN, 1996]:

> A Business Process is a partially ordered set of Enterprise Activities which can be executed to realize a given objective of an enterprise or part of an enterprise to achieve some desired end result.

Whereas other enterprise models concentrate on functional decomposition and modeling views, an executable model specifies business processes as chains of enterprise activities.

Business processes and enterprise activities are triggered by events. Event Management is thus one of the MXS services. Event Management issues requests to the Business Process Management

Service for executing business processes. Since each business process is a chain of enterprise activities, Business Process Management relies on the Enterprise Activity Management Service for the execution of individual activities. Enterprise Activity Management requests resources and information from the Resource Management Service and the Information Management Service, respectively. For each activity, the information is presented to the assigned resource by means of the Presentation Management Service. A Run-time Repository service tracks all information concerning the execution of business processes and enterprise activities. Figure 4.11 illustrates the relationships between these services (the Run-time Repository is not shown).

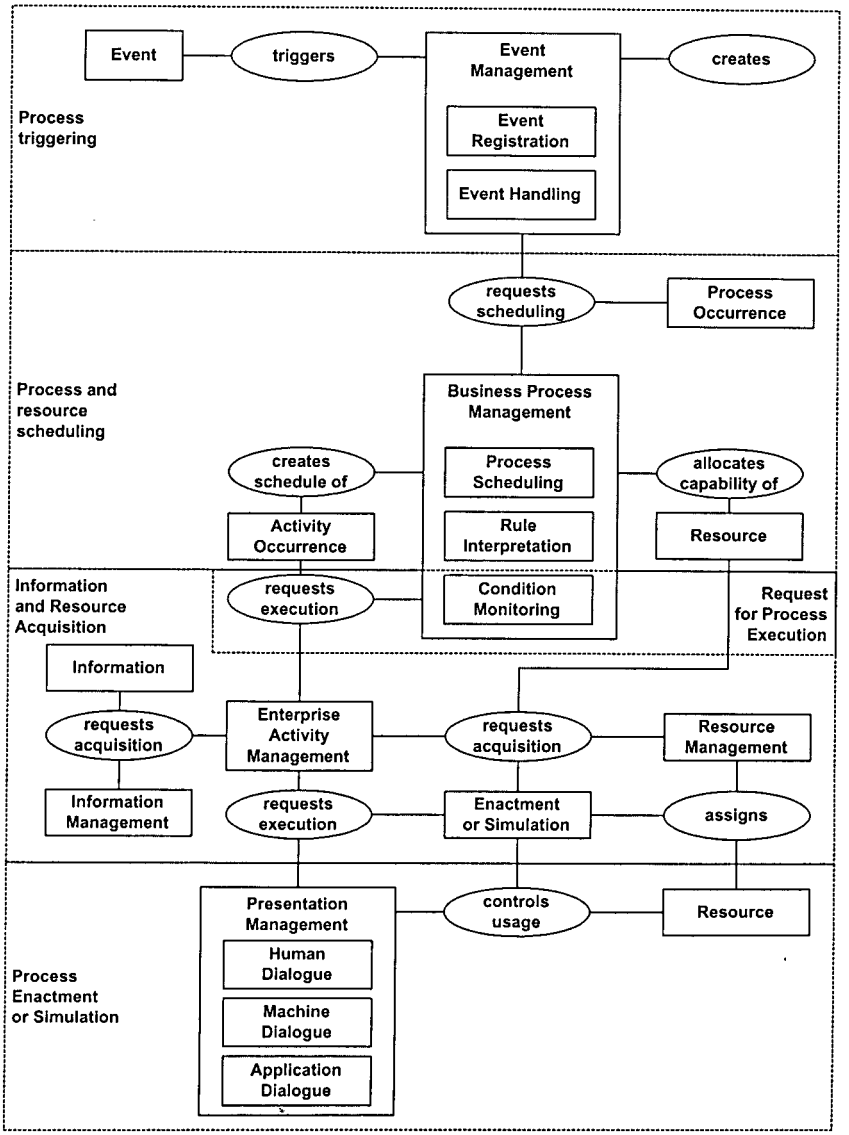


Figure 4.11: EMEIS phases of model execution[69]

[69][CEN, 1998]

External events with information items such as orders are used to initiate business processes. Within business processes, process events denote the beginning and completion of enterprise activities. Event Management is divided into functionality belonging to Event Registration and that belonging to Event Handling. Event Registration records events and registers which business processes and enterprise activities are triggered by those events. Upon receipt of an event, Event Handling requests the scheduling of the business process that is triggered by that event, creating a new process occurrence that will be interpreted and monitored by the Business Process Management Service.

The Business Process Management Service retrieves the business process definition from the run-time repository and invokes the Resource Management Service to allocate the required resource capabilities for that process, by means of its process scheduling functionality. This functionality also requests the Event Registration to include the possible events generated within the new process occurrence. Every time such an event occurs, Rule Interpretation is invoked to finalize or trigger enterprise activities as appropriate. For this purpose, Rule Interpretation requests Condition Monitoring functionality to periodically evaluate the conditions and constraints of behavioral rules. Condition Monitoring notifies Rule Interpretation of conditions that become true or constraints that have been infringed.

The Enterprise Activity Management Service is responsible for initiating enterprise activities according to the results of Rule Interpretation. For each activity, the Enterprise Activity Management Service requests Resource Management to assign resources allocated by Process Scheduling. At the same time, the Enterprise Activity Management Service invokes Information Management in order to acquire enterprise information objects that will be used as functions and controls inputs for the activity being performed, as depicted in phase 3 of figure 4.10. As soon as both the assignment of resources and the acquisition of information are complete, the Enterprise Activity Management Service invokes the Presentation Management Service so as to carry out the execution of the enterprise activity.

The Presentation Management Service provides the appropriate interface for presenting information to the assigned resources during the execution of an enterprise activity. EMEIS assumes that there are three particular specialization of resources that have to be treated differently at run-time, because there are different ways to invoke them and different ways for them to return results. According to EMEIS, resources are classified as human, machine or application resources. Consequently, the functionality of the Presentation Management Service is divided, as in CIMOSA, into Human Dialogue, Machine Dialogue and Application Dialogue. The Application Dialogue supports data exchange between MXS and software applications, and promotes the availability and use of these applications in the workplace. The Machine Dialogue provides access to functional capabilities of machine resources through device drivers and is able to receive and interpret returned results. The Human Dialogue supports, in addition to the exchange of information related to human tasks, a user front-end facility to models and their run-time environment, it allows users to monitor business processes, handle exceptions, browsing resources, visualizing models according to different views, and exploring alternative scenarios by means of simulation.

The Resource Management Service responds to requests from Process Scheduling to pre-allocate the capabilities required for each business process occurrence. The Resource Management Service checks the availability of resources and their information inputs and pre-assigns them when available, signaling to Process Scheduling that those resources are reserved. The pre-assigned resources will be requested for action every time their corresponding activities are brought into execution by the Enterprise Activity Management Service. The Resource Management Service will release the allocated resources when the process occurrence is complete.

The Information Management Service provides access to system-wide information such as administrative, technical and multimedia documents, as well as to EDI facilities. EMEIS requires this service to perform conversion between data formats and to give transparent access to data stored in several, heterogeneous database systems. This service should provide transparency regarding data location, replication and storage means. It should also support the flexible change of enterprise information, including the definition of data formats, version control of those formats, and enforcement of data access privileges.

The Run-time Repository serves as a registry for all other services to maintain information regarding the execution and simulation environments of business processes. The Run-time Repository maintains the register of business processes, enterprise activities, their sequencing and conditional rules, their triggering events, and their conditions and monitoring frequencies for Condition Monitoring. The Run-time Repository also keeps general system configuration.

4.3.3 Shared and Base IT services

EMEIS realizes that both MDS and MXS require a set of services - referred to as Shared Services - that include:

- reliability services - include functionality for monitoring internal queues and data structures of the execution services; also report delays in processing events or in allocating resources, while responding to these delays with load-balancing or by raising exceptions;
- security services - assign and manage user privileges defining allowable operations and the degree of access to models, authenticate user identities, and control access to applications;
- contract and broking services - (the name is perhaps misleading) these services allow the discovery and invocation of other service capabilities such as encapsulated legacy systems;
- encapsulation services - provide application wrappers for legacy systems and applications, encapsulating their functionality as service capabilities and defining their invocation and the range of possible responses;
- distribution services - comprise communication protocols to develop, exchange and execute models over more than one execution platform, while ensuring a common, system-wide encoding of model semantics.

On a lower level under these services, EMEIS places a set of Base IT Services that cope with the heterogeneity and distribution of model development and execution environments. The Base IT Services should enable systems interoperability by providing fundamental functionality such as naming, sharing, and protection of distributed model components. These components may run on platforms that require different access protocols. Base IT Services should also support the synchronization, consistency and error recovery of model components running in different platforms, and should employ version control of models and integrity of model components developed independently, at different times or sites.

4.4 Towards an inter-enterprise integration architecture

The previous sections have presented an overview of the major enterprise integration reference architectures that address the development of enterprise models and the need for integration infrastructures.

From CIMOSA, GRAI-GIM and PERA, to GERAM and EMEIS, the discipline of Enterprise Integration has been in pursuit of concepts, methodologies and architectures with increasing generality and applicability. In the beginning, Enterprise Integration aimed at supporting the design of CIM systems, which is illustrated by the development of CIMOSA. However, recognizing that today's enterprises are facing rapidly changing environments, Enterprise Integration has evolved towards a discipline that organizes the knowledge and methods required to identify and cope with change. This discipline is called *enterprise engineering* [Vernadat, 1996], and it is about designing or reengineering enterprises in a systematic way using some structured approach and computer-supported tools similar to those used in software engineering.

This trend would eventually lead to GERAM, which "is about those methods, models and tools required to build and maintain the integrated enterprise, be it a part of an enterprise, a single enterprise, or a network of enterprises" [IFIP-IFAC, 1999]. It can therefore be argued that this work is related to Enterprise Integration in two ways. On one hand, business networking is an enterprise engineering problem that requires a systematic approach to the integration of networks of enterprises. On the other hand, workflow management systems are the engineering tool that will allow enterprises to build and operate business networks. The following sections explain the relation of Enterprise Integration to workflow management and business networking in more detail.

4.4.1 Workflow Management and Enterprise Integration

Within the discipline of Enterprise Integration, CIMOSA deserves special attention. Being the first major research initiative in Enterprise Integration, CIMOSA was also the first to establish a set of concepts (such as that of Business Process) and a model-based integration approach that would shape following integration architectures, if not the whole field of Enterprise Integration. First, the CIMOSA modeling framework depicted in figure 4.2 is still perhaps the most comprehensive and structured approach for the development of enterprise models. Second, the distinction and relationship between the engineering and operation environments shown in figure 4.4 is the basis for integration architectures that support both the development and operation of the integrated enterprise. And third, CIMOSA was the first integration architecture to explicitly recognize the need for an integration infrastructure comprising several services, as depicted in figure 4.5. Remarkably, its Communication Services layer, which provides features such as naming services, message queuing, location transparency, replication and concurrency, seems premonitory when compared to distributed middleware architectures available today, such as CORBA. In addition, the CIMOSA Integrating Infrastructure introduced the innovative possibility of executing enterprise models, and its Business Services layer is astonishingly similar, both in purpose and operation, to workflow enactment services.

In fact, most workflow concepts can be mapped onto CIMOSA concepts and terminology [Dickerhof et al., 1999]. Furthermore, the WfMC's interface 1 (process modeling), interface 2 (workflow client applications) and interface 5 (administration and monitoring) correspond to model generation, Presentation Services and System Management Services in CIMOSA, respectively. The worklist of a workflow participant, for example, is equivalent to the Human Dialogue in CIMOSA. However, the modeling framework of CIMOSA clearly surpasses the modeling scope of workflow management systems. Whereas CIMOSA addresses the function, information, resource and organization views, workflow process definitions address mainly the function view and a limited information view, with implicit assumptions on resource and organization views. Nevertheless, within certain restrictions it has been shown that it is possible to implement CIMOSA models using a workflow management system [Ortiz et al., 1999].

Workflow management is related to other enterprise integration architectures as well. In the Carnot project [Huhns et al., 1992], which aims at unifying heterogeneous information resources, database queries are expressed according to task graphs [Tomlinson et al., 1993]. These task graphs are expanded into a set of local queries that include access to the semantically equivalent, relevant data sources. Furthermore, the task graph is augmented with additional sub-tasks that must be performed in order to maintain enterprise-wide data consistency. The resulting task graph is equivalent to a workflow process definition which comprises a set of application invocation activities.

In ARIS [Scheer, 1992], workflow management supports the enterprise modeling level. Event-driven process chains may be used as process definitions, while the remaining ARIS views are used to check the consistency of process models. Then, through a specific procedure, as described in [Kronz, 2001], ARIS models may be implemented using a third-party workflow management system.

Within GERAM, workflow management can have a crucial role if used as the Methodology Entity (entity type 5), as illustrated in figure 4.8. In this case, it supports the life cycles of the Engineering Implementation Entity (entity type 2) and of the Enterprise Entity (entity type 3). In addition, the functionality of workflow management systems falls within the scope of several GERAM framework components. As shown in figure 4.12, the ability to model and analyze business processes suggests that a workflow management system can be used as an Enterprise Engineering Tool that supports the development of Enterprise Models, and the ability to execute business processes and to interact with resources suggests that workflow management systems are indeed a kind of Enterprise Operational Systems.

Positioned within the framework of EMEIS, it can be seen that (1) workflow management systems span across Model Development Services and Model Execution Services, and that (2) they require Shared and Base IT Services, i.e., they require an integration infrastructure that allows all resources to interoperate with each other.

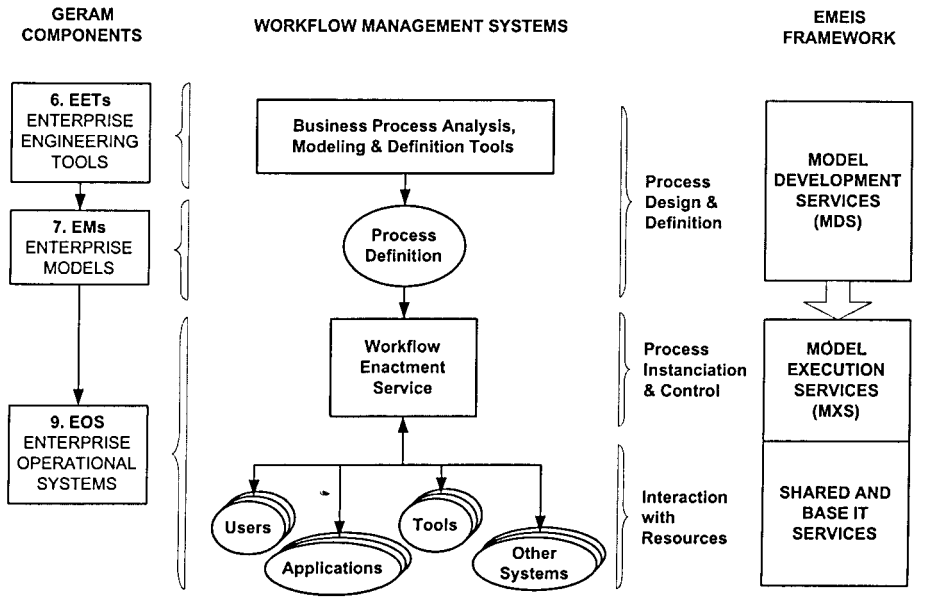


Figure 4.12: Workflow Management vs. GERAM and EMEIS

4.4.2 Business networking and Enterprise Integration

The GERAM integration architecture can be given a special meaning in connection with business networking, due to the fact that its entities can be mapped directly onto several facets of a business network. The Strategic Management Entity (entity type 1) deals with a set of business goals and identifies the market opportunities that require the linkage of several competencies into a business network. The Engineering Implementation Entity (entity type 2) is the entity which provides the means for an enterprise to integrate itself with other enterprises, and which carries out that integration effort; this entity may or may not belong to the enterprise. The Enterprise Entity (entity type 3) is the result of that integration effort, i.e., it represents the way an enterprise will interact and operate with other enterprises within a business network. The Product Entity (entity type 4) corresponds to the ultimate product or service produced by a business network. The Methodology Entity (entity type 5) is, in the framework of this work, an integration approach based on workflow management.

The way life cycles are connected to each other in GERAM can be regarded as a generalization of a concept that was first introduced by CIMOSA: the relationship between the system life cycle and the product life cycle, shown in figure 4.4 on page 144. GERAM extends this concept to the relationship between the life cycles of four of its five entity types, with the fifth entity representing the methodology that supports the operation of both entity type 2 (the Engineering Implementation Entity) and entity type 3 (the Enterprise Entity), as suggested in figure 4.8 on page 150. This means that workflow management, as a tool that provides process modeling and execution capabilities, can support not only the operation of a business network (which takes place during the operation of entity type 3), but also the design and execution of the business processes that control the development of the business network itself (this design takes place during the operation of entity type 2). Therefore, enterprise integration brings a new dimension where workflow management systems can be applied: it is not only a matter of coordinating inter-enterprise business processes (equivalent to CIMOSA's operation environment), but also of supporting the requirements, design and implementation of those processes (the engineering environment).

The challenge of coordinating inter-enterprise business processes can be regarded as the challenge of supporting an operational life cycle (the life cycle of entity type 4), when each member of a business network performs a certain task according to its own competence. On the other hand, the challenge of defining how an enterprise searches for appropriate business partners, how it selects them and how it establishes which tasks they will perform, belongs to an engineering life cycle (entity type 3), so the operational life cycle is just a single phase within an engineering life cycle. This results in a relationship between the operational life cycles of several enterprises, as suggested in figure 4.13.

If this operational life cycle can be described by means of a business process model, then it is possible to automate this life cycle by means of workflow management, and therefore to automate the whole life cycle of a business network, from the point when an enterprise searches for business partners to the point when they complete the business relationships that bind them together. From the point of view of business networking, it is useful to further separate the enterprise operational life cycle into two components: one is a kind of engineering process, which defines how an enterprise searches for business partners and develops business relationships with them; the other is a kind of operational process which, running within the operation phase of that engineering process, takes place when a set of predetermined business partners interact with each other in order to provide a product or service to an end-customer. The workflow management system that allows both of these processes to be systematically modeled and executed, and then redesigned and executed again as many times as necessary, is a workflow management system that supports the engineering of business networks.

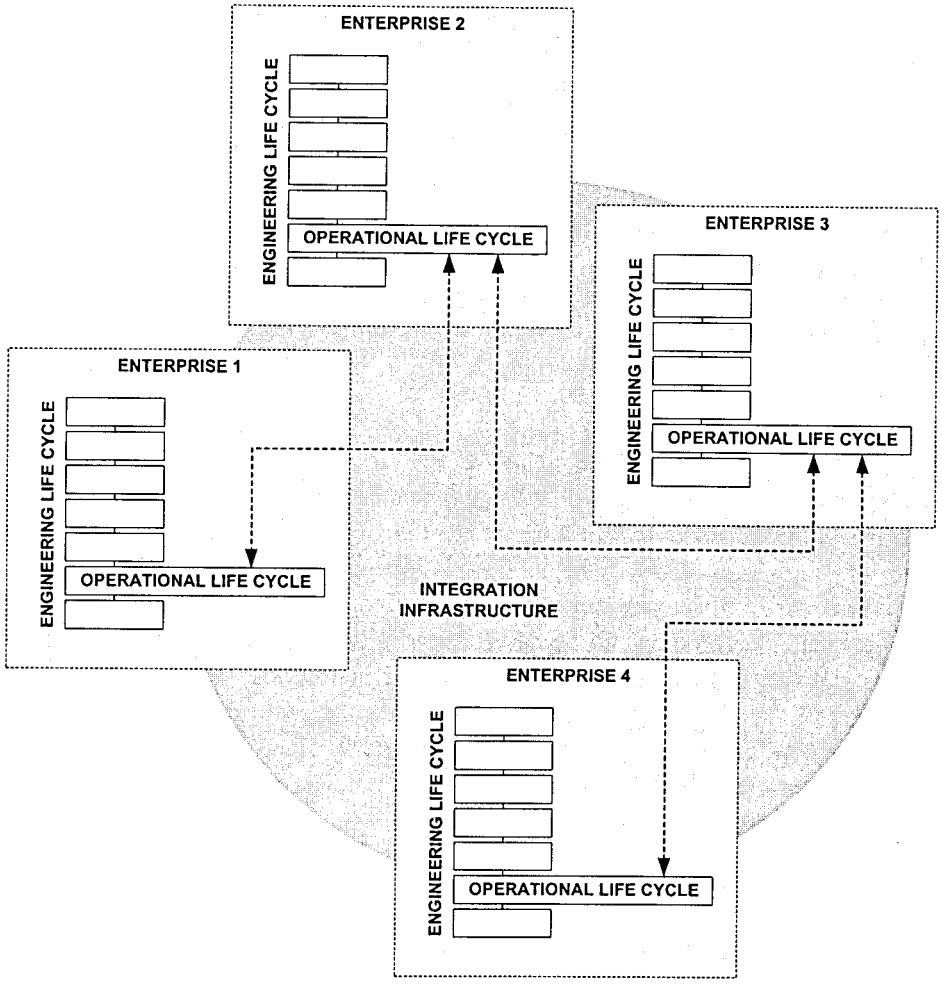


Figure 4.13: Relationships between enterprise life cycles within a business network

4.4.3 The B2B trading life cycle

Figure 4.13 suggests that a business network is a combination of several business relationships running in parallel between different pairs of enterprises. But since enterprises are autonomous entities and are free to establish relationships with any business partners, each enterprise has control over the development of its relationships with other enterprises. Therefore, a workflow management system that supports the engineering of business networks cannot be devised as a centralized point of control over the actions of several enterprises. Instead, a workflow management system that supports the engineering of business networks must provide workflow functionality that is replicated across several enterprises and that, at each enterprise, supports the life cycle of business relationships with other enterprises.

Each business relationship is a relationship between two enterprises that first find each other, then interact with each other, and afterwards evaluate the performance of each other. But referring to the life cycle of a business relationship is the same as referring to the life cycle of a Business-to-Business (B2B) trading relationship, in which one enterprise plays the role of buyer and the other enterprise

plays the role of a supplier. Therefore, the life cycle of a business relationship can be regarded as being comprised of five main phases (this life cycle is equivalent to that presented in [Piccinelli et al., 2001]):

1. *The search phase* - The life cycle begins at the identification of some market opportunity or purchase need. The buyer will conduct a survey on the market in order to collect information about a list of potential suppliers. During the search phase, the main challenge is how to advertise selling offers, purchase needs, and how to match these elements against each other. This is not a problem of simply defining a document format; instead, it is a problem of defining what information should be advertised and how should it be conveyed. Buyers must advertise purchase needs in such a way that it can be matched with the available selling offers. If each buyer or supplier develops its own method of advertising selling offers and purchase needs, it will be very difficult to match and identify trading opportunities.

2. *The selection phase* - The search phase may produce some results that do not address exactly the original purchase need, so these entries should be pruned by means of a pre-selection of the relevant suppliers. Once there is a set of one or more suppliers that are able to fulfill the purchase need, the buyer will engage in one-to-one conversations with those suppliers. In each conversation, buyer and supplier exchange information about the desired products, quantities, delivery method, price, and payment method in order to ascertain the conditions for the purchase of goods, services, or technology. Once the results from these inquiries have been collected from all of the prospective suppliers, the buyer will be able to identify the supplier providing the best conditions, by considering parameters such as price, quality and service fulfillment.

3. *The contracting phase* - The purpose of the contracting phase is to elaborate and sign a contract or Trading Partner Agreement (TPA) between buyer and supplier. Basically, the TPA specifies how the purchase will take place, from initial ordering to final payment. For a workflow management system supporting the engineering of business networks, it is especially important that the TPA specifies the operational process that will govern the interaction between the two enterprises during the operation phase. Each of the parties that sign a TPA may also establish TPAs with other business partners that provide specialized services. The set of all TPAs that are developed in order to react to a market opportunity define the business relationships within a business network.

4. *The operation phase* - During the operation phase, buyer and supplier perform the exchanges they have previously contracted. As will be described in the next chapter, there are several B2B frameworks that address the operation phase. These B2B frameworks, which are mostly XML- and EDI-based specifications, define message structures and document formats for a wide range of B2B exchanges such as product information, ordering, customer service and financial exchanges. Therefore, even though there may be no B2B framework that exactly fits the requirements for a given buyer-supplier relationship, there is no need to develop proprietary exchange formats. The amount of effort that has been put on the development of B2B frameworks suggests that it should be possible to reuse suitable document formats or to adapt them to particular exchange needs.

5. *The evaluation phase* - In a final evaluation phase, the buyer will measure the performance of the supplier and this information can be used as input in future partner selections. Examples

of key performance indicators are task execution time, quality/price relation and effectiveness of customer service. Besides collecting information about the capabilities and performance of the contracted supplier, the buyer will periodically probe the market in order to determine the capabilities and performance of other suppliers of the same products or services, and in order to become aware of new suppliers that may have entered the market. On the other hand, the dynamics and competitiveness of the market will require suppliers to continuously improve their operational processes, and to make every effort to expand their customer base.

This B2B trading life cycle describes one possible way to develop pairwise business relationships, which are the building blocks of a business network. Therefore, from the point of view of business networking, this B2B trading life cycle could have been called the engineering life cycle of a business network. However, since in general there can be no centralized point of control over the actions of several autonomous enterprises, the engineering life cycle of a business network is actually scattered over several enterprises. Each of these enterprises carries out these five phases independently of other enterprises, although connected with at least some of them. In particular, the operation phase of the B2B trading life cycle is equivalent to what could accordingly have been called the operational life cycle of the business network, since, at this point, its members have already established a set of business relationships, and now they will interact in order to perform the tasks that have been assigned to each one of them.

To make these concepts clearer, figure 4.14 illustrates the relation between the life cycles of an enterprise and those of a business network. The enterprise operational life cycle, during which the B2B trading life cycle is carried out, is equivalent to the idea of a business network engineering life cycle. On the other hand, the operation phase of the B2B trading life cycle is equivalent to the idea

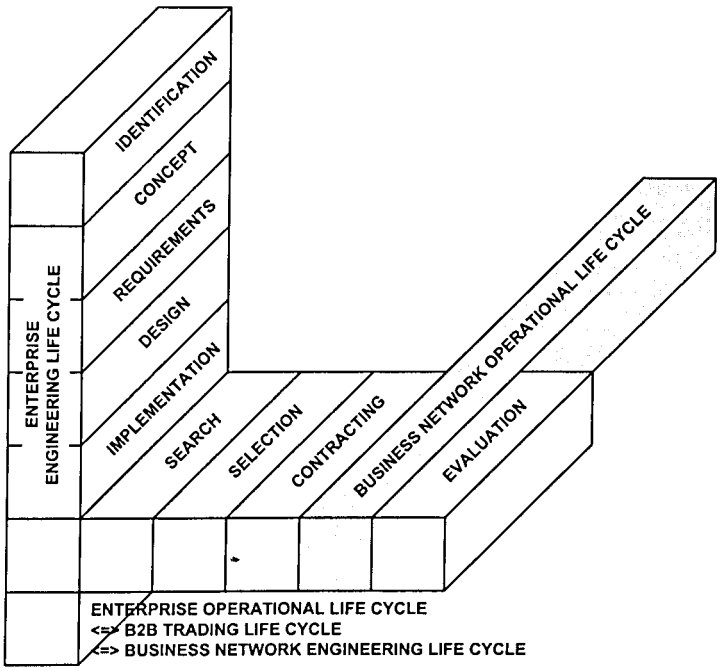


Figure 4.14: Relation between enterprise and business network life cycles

of a business network operational life cycle, which runs within the operation phase of the business network engineering life cycle. Once again, it should be noted that the business network engineering life cycle is scattered over several enterprises and therefore cannot be represented by means of a single process definition. In subsequent chapters, particularly in chapter 7, the business network engineering life cycle, when expressed as a set of workflow processes to be released for execution at different enterprises, will be often referred to simply as "the engineering process".

4.4.4 The enterprise engineering life-cycle

The life cycle described in the previous section is just one possible way in which enterprises can establish relationships with each other. Enterprises could as well develop their relationships according to a different B2B trading life cycle, which could comprise more, less or just different steps. The advantage of using workflow management to support this or any other engineering life cycle is that these steps can be modeled as a workflow process and then the process definition can be released for automated execution. In addition, should the process change or be improved, the process definition can be modified and released for execution again. Therefore, a workflow management system that automates the whole life cycle of business relationships is in effect an enterprise engineering tool that supports some of the life-cycle phases of the enterprise itself, as suggested in figure 4.15.

The B2B trading life cycle, such as the one described in the previous section, is modeled during the requirements, design and implementation phases, and it runs within the operation phase of the enterprise engineering life cycle. This process model could be developed, for example, as suggested in [Vernadat, 1996]: during the requirements definition phase the process is modeled or redefined, possibly as a set of partial models, and checked for consistency; during the design specification phase the model is simulated or animated using Petri nets, and its performance is evaluated; during the implementation description phase the enactment system is configured for model execution; finally, during the operation phase the model is released for execution over an integration infrastructure.

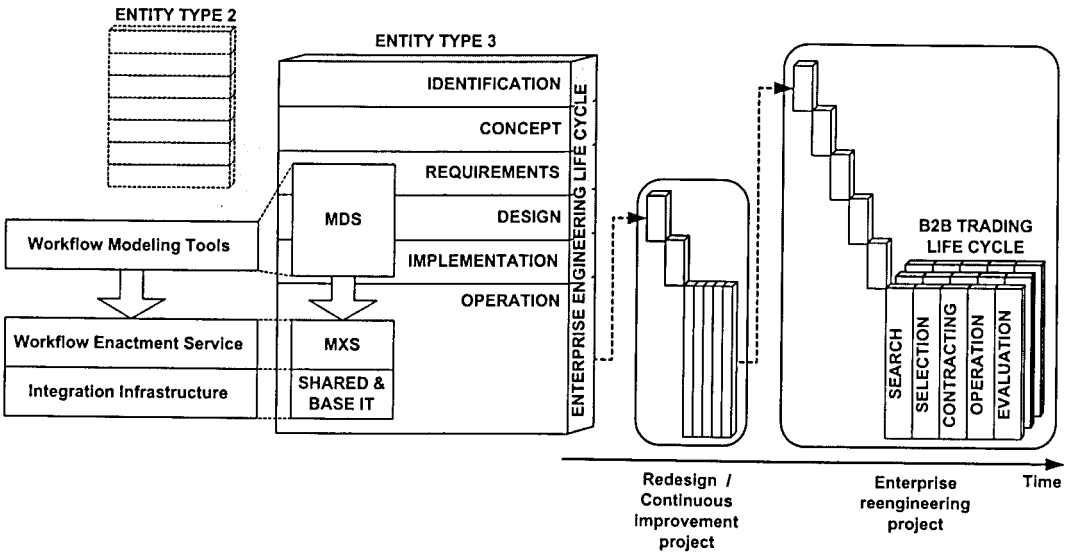


Figure 4.15: The enterprise life history

4.4.5 Matchmaking approaches

Whereas the selection, contracting, operation and evaluation phases of the B2B trading life cycle could possibly be carried out as ordinary workflow activities, the search phase is more difficult to deal with since it requires special facilities. A workflow management system that is intended to support the engineering of business networks must provide the ability to discover new business partners, and the ability to integrate business processes with any potential business partner. Some workflow management systems have already attempted to cope with these requirements: ACE-Flow (section 3.5.3.4) and WISE (section 3.5.3.6) have devised catalogs that allow enterprises to advertise their services, which become building blocks of inter-enterprise business processes; COSMOS (section 3.5.3.7) and CrossFlow (section 3.5.3.8) make use of catalogs that contain contract templates, which describe how inter-enterprise business processes should be executed. However, these solutions are highly centralized.

In fact, there is a general trend towards partner search solutions based on centralized repositories of information, and this trend is not restricted to workflow management systems. An example is Universal Description, Discovery and Integration (UDDI), which will be presented in the next chapter, and which defines a repository for enterprises to store information about the Web Services they implement. The repository contains information about service providers (which play an equivalent role to suppliers), and this information is to be retrieved by service clients (equivalent to buyers) as they search for enterprises that implement a desired service. Thus, in this case, it is the service client who tries to match its own needs to the offers available in the repository.

In other approaches, both suppliers and buyers store their offers and needs in a repository so that they can be matched to each other. The matchmaking can be performed by buyers and suppliers themselves, as they retrieve information about one another from the repository. Such is the case of e-marketplaces, which provide a meeting point on the Web for buyers and suppliers, as explained in section 2.2.4.3. Alternatively, the matchmaking may be performed by a third-party that keeps searching for business opportunities between supplier offers and buyer needs. Such is the case of the Innovation Relay Centre Network, which comprises several organizations that cooperate closely in order to encourage technology transfer within Europe, as described in appendix A.

4.5 Concluding remarks

The development of business relationships between enterprises takes place within a B2B trading life cycle which, when expressed as a set of workflow processes to be released for execution at different enterprises, can be collectively regarded as the engineering process of a business network. Enterprises must be provided with a solution to design and control this engineering process. On one hand, they need process modeling and execution capabilities to describe and execute this process as an interaction between resources that belong to different enterprises. On the other hand, enterprises also need an integration infrastructure that allows those resources to interoperate across enterprise boundaries. The actions of these resources must be coordinated so that, even though they are under the control of different enterprises, their overall behavior fits into the business process running at each enterprise. Workflow management is fundamental to achieve this kind of coordination, and having an appropriate integration infrastructure, as suggested in figure 4.13, is fundamental to apply workflow management in an inter-enterprise environment.

The resulting integration architecture should exhibit the following characteristics, which apply to any integration architecture in general [Vernadat, 1996]: it should be *open*, it should be *expand-*

able, it should be *modular*, its components should be *reusable*, and it should cater for *cooperative* solutions. In addition, an inter-enterprise environment requires that, in order to safeguard the autonomy of each enterprise and the possibility of establishing business relationships with any partner, the integration architecture should be *decentralized*. On one hand, this means that there should be no centralized workflow control over the members of a business network; each enterprise should have its own workflow enactment service and the workflow enactment services at different enterprises should be configured in order to implement the desired business relationships. On the other hand, it means that the underlying integration infrastructure itself must be decentralized so that no single enterprise can own or control the services that allow enterprises to interact with each other.

The previous chapter had already concluded that there is a duality between process modeling/execution and the underlying integration infrastructure, and that workflow management systems address mainly the first part. Enterprise integration architectures bring additional opportunities to apply workflow management systems, and they introduce the need for integration infrastructures to facilitate information exchange and interaction between resources. Therefore, concerning both Model Development Services (MDS) and Model Execution Services (MXS), as defined by EMEIS, the previous chapter has shown that there are many workflow solutions available. But regarding Shared Services and Base IT services, which concern the underlying integration infrastructure, EMEIS requires them to enable interoperability between different systems and to provide several other run-time capabilities, and GERAM goes even further to require an integration infrastructure across heterogeneous enterprise environments [IFIP-IFAC, 1999]. The next chapter presents an overview of the technologies that are most relevant to the development of such an inter-enterprise integration infrastructure.

Chapter 5

Infrastructures for B2B Interoperability

The previous chapter has introduced a broader perspective on the problem of integration, particularly on business process integration. From the set of all integration architectures developed within the field of Enterprise Integration, two results are especially important. One was the development of a generalized integration architecture - GERAM - which defines a set of entities and their life cycles, as well as the possibility of establishing relationships between the life cycles of different entities. The other important result was the development of a generic framework - EMEIS - which describes the kinds of services that are required for any enterprise integration approach. These services must support the development and execution of enterprise models, and they must cope with requirements such as reliability, security, distribution and encapsulation.

Given that the proposed approach to the integration of business networks is based on workflow management, figure 5.1 shows how workflow management systems could be positioned within the framework of EMEIS. The fact is that existing workflow management systems provide most Model Development Services and Model Execution Services. However, regarding Shared and Base IT Services, i.e., regarding the underlying integration infrastructure, it becomes apparent that most workflow management systems do not provide the same amount of functionality as they do regarding process modeling and execution capabilities. This can be explained by the fact that existing workflow management systems are unaware of the duality between the workflow enactment service and its underlying integration infrastructure, and of the fact that these components should be developed separately.

As chapters 3 and 4 have shown, the challenge of coming up with an appropriate integration infrastructure must be handled as an equally important problem as that of devising process modeling and execution functionality. Therefore, the development of an infrastructure for the integration of business networks requires a study of the approaches and technologies that are relevant to that purpose. In this work, four technologies have been considered: message-oriented middleware, business-to-business frameworks, intelligent software agents, and Web services. The reason for having considered these technologies is that they seem to be able to cope with different infrastructural requirements, as suggested in figure 5.1, although this is only an approximate assessment.

The previous chapter had concluded that such an integration infrastructure should have an important characteristic: it should be a decentralized platform. For this reason, a fifth technology has been considered besides the technologies mentioned in figure 5.1. The additional technology is peer-to-peer networking, which promotes completely decentralized solutions. This technology does not appear in figure 5.1 because, so far, it has been used in very particular situations, so it lacks the services that would allow it to be applied in business environments. Nevertheless, it is thought that

		Workflow Management Systems				Infrastructures for B2B Interoperability			
		Commercial Workflow Products	System-centered Workflow	Internet-mediated Workflow	Inter-enterprise Workflow	Message-Oriented Middleware	B2B Frameworks	Agent-based systems	Web Services
Model Development Services (MDS)	Model Creation	x	x	x	x				
	Model Assessment	x	x						
	Model Management		x		x		x		x
	Model Repository				x		x		x
Model Execution Services (MXS)	Event Management	x	x	x	x				
	Business Process Mgt.	x	x	x	x				
	Enterprise Activity Mgt.	x	x	x	x				
	Presentation Mgt.	x	x	x	x				
	Resource Mgt.								
	Information Mgt.	x							
	Run-time Repository	x	x						
Shared and Base IT Services	Reliability	x	x			x			
	Security	x				x	x		
	Contract and broking				x		x	x	x
	Encapsulation	x						x	x
	Distribution			x				x	

Figure 5.1: Integration technologies in the framework of EMEIS

peer-to-peer networking has the potential to become an equally important technology for the development of inter-enterprise integration infrastructures.

The following sections provide an overview of these five technologies in order to ascertain how each of these technologies could be used to promote interoperability between enterprises, and, ultimately, to assess the potential of these technologies to become the underlying integration infrastructure for a workflow management system that supports the engineering of business networks.

5.1 Message oriented middleware (MOM)

Message-oriented middleware (MOM) [Banavar et al., 1999] is a client/server infrastructure that supports the exchange of general-purpose messages in a distributed application environment. Typically, these messages are exchanged asynchronously between applications: the message is sent by one application and received by another, but these events may happen at different times. One of the distinctive features of MOM is that it provides temporary storage when the destination application is busy or not connected. Another distinctive feature of MOM is that it provides a single interface for every application to send and receive messages. Therefore, using this interface is enough to be able to communicate with every other application integrated with the MOM system. However, applications have to semantically understand each other in order to exchange information, which is not addressed by MOM. A MOM system provides message-based communication between applications, regardless of the content of messages.

Due to its distinctive features, MOM facilitates the interoperability and flexibility of applications. It reduces the complexity of integrating new applications with existing ones, and it is able to integrate applications over heterogeneous platforms. MOM systems are most appropriate for event-driven applications and applications with long transaction lifetimes such as, incidentally, workflow management systems. MOM exhibits a number of advantages. One such advantages is that applications are free to perform operations while waiting for a response from another application. Another advantage is that MOM allows many responses to one request, and many requests to one response.

With additional functionality, MOM systems are able to implement message priority, load balancing, fault tolerance, and persistent messages.

There are three main types of MOM systems: message passing, message queuing, and publish/subscribe systems. In all these systems, communicating applications behave as clients of a central messaging server, as shown in figure 5.2. In message passing systems there is direct, synchronous communication once both applications establish a connection to the MOM system. On the other hand, message queuing systems are connectionless, and communication is asynchronous because the message is not directly sent to the destination application; instead, the message is kept on a queue until the destination application is able to read it. While the message is kept in the queue, both applications are free to perform other operations. The third type of MOM system is publish/subscribe, where each application has its own message queue. In this case, messages are sent without a specific destination, and it is up to the MOM system to decide which application will receive the message. For a given published message there may be any number of interested applications (subscribers), hence the name of publish/subscribe. This kind of MOM system is the most flexible since applications may be dynamically configured as publishers or subscribers, effectively changing the route of messages and the interaction between applications.

Most of the currently available MOM systems support these three types of message exchange. But MOM systems have been typically implemented as proprietary products, which means that MOM implementations, in general, cannot be expected to interoperate with each other. However, MOM vendors have been increasingly adopting standard messaging interfaces, especially the Java Message Service (JMS) API [Sun, 2002]. Even though internal implementation details differ from MOM

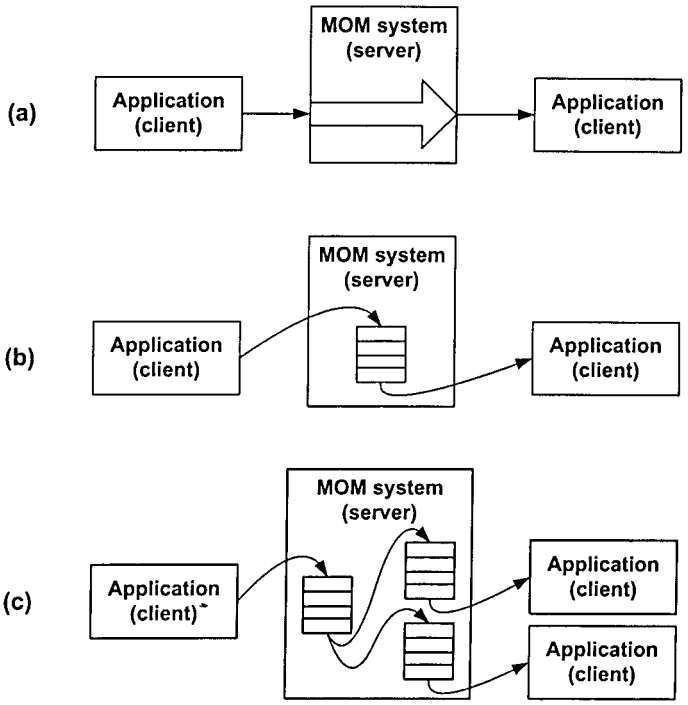


Figure 5.2: Message passing (a), message queuing (b) and publish/subscribe (c)

system to MOM system, most of these systems are now able to expose a common interface to applications. Thus, it is likely that MOM systems will become interoperable in the near future. Figure 5.3 presents a brief survey of MOM products, where it can be seen that most products already support the JMS API, i.e., their internal implementation is accessible through a JMS-compliant interface. Also shown in figure 5.3 is the fact that some of these MOM systems are effectively used as the underlying infrastructure for workflow management systems. For example, MQSeries is the underlying communication infrastructure for the MQSeries Workflow system provided by the same vendor, and TIBCO Rendezvous can be used together with TIBCO InConcert workflow system. Moreover, MOM systems are usually designed to support legacy systems and widely dispersed systems.

5.1.1 The Java Message Service (JMS)

The Java Message Service (JMS) [Sun, 2002] defines a standard, object-oriented API for MOM systems. JMS supports two messaging styles - point-to-point and publish/subscribe - which are equivalent to cases (b) and (c) in figure 5.2, respectively. For point-to-point communication, JMS specifies a set of interfaces to deal with message queues, and for publish/subscribe communication it provides another set of interfaces to deal with message topics. Both these sets of interfaces share the concepts of connection (to the MOM system), destination (queue or topic), session (a delimited messaging interaction), message producer (used for sending messages), and message consumer (used for receiving messages).

Basically, a JMS client first establishes a connection to the MOM system, and then uses the connection to create one or more JMS sessions. Within a session the JMS client can create several destinations, including queues (for point-to-point communication) and topics (for publish/subscribe messaging). The client will use the session and the destination objects to create message producers and message consumers as needed by the application. Finally, the client can request message pro-

Vendor	URL	Product	Open source	Supports JMS	Used together with workflow
4 Tier Software	http://www.xing.com/	OpenMOM			
Ashnasoft	http://www.ashnasoft.com/	AshnaMQ		x	
BEA Systems	http://www.beasys.com/	MessageQ			
ExoLab Group	http://www.exolab.org/	OpenJMS	x	x	
Fiorano	http://www.fiorano.com/	FioranoMQ		x	x
IBM	http://www.ibm.com/	MQSeries		x	x
JBoss.org	http://www.jboss.org/	JBossMQ	x	x	
Level8	http://www.level8.com/	Geneva Integration Broker			x
Microsoft	http://www.microsoft.com/	MSMQ			x
my-Channels	http://www.my-channels.com/	Nirvana		x	
ObjectWeb	http://www.objectweb.org/	JORAM	x	x	
Oracle	http://www.oracle.com/	Advanced Queueing		x	x
Softwired	http://www.softwired-inc.com/	iBus Message Server		x	
Sonic Software	http://www.sonicsoftware.com/	SonicMQ		x	
Sun Microsystems	http://www.sun.com/	ONE Message Queue		x	
Talarian	http://www.talarian.com/	SmartSockets		x	
TIBCO	http://www.tibco.com/	Rendezvous		x	x
Vertex Interactive	http://www.vertexinteractive.com/	NetWeave			
xmlBlaster.org	http://www.xmlblaster.org/	xmlBlaster	x		

Figure 5.3: Overview of MOM system vendors and products (August 2002)

ducers to send messages, and message consumers to receive them. This behavior is the same both in point-to-point and publish/subscribe messaging styles, and it is illustrated in figure 5.4.

The JMS specification includes support for reply/request messaging interactions, since it is one of the most common communication patterns between applications. For this purpose, JMS reserves a special message header field to specify the reply destination for each message. In addition, JMS allows the creation of temporary queues and topics that can be used as unique destinations for message replies, both in point-to-point and publish/subscribe operation. In fact, JMS even specifies the implementation of basic functionality to support request/reply behavior. It is left up to the MOM system to provide more specialized implementations.

In general, JMS specifies only a set of interfaces that the MOM system must implement somehow. The MOM system is free to devise its own implementation of these interfaces, and the implementation of additional functionality as well. The JMS specification is essentially an abstraction of those messaging features that are common to all MOM systems. For this reason, JMS does not address sensitive issues such as security (message integrity and privacy), load balancing, fault tolerance, system errors, system administration, or message formats.

5.1.1.1 The message model

JMS does, however, specify its own message model. According to JMS, each message consists of a header, a set of properties, and a body. The header contains fields used by clients and the MOM system for message routing and identification. These fields include:

- the destination to which the message should be sent, expressed as an object reference;
- the delivery mode, which is used to specify if the message should be made persistent or not;
- a message identifier to be used as a unique key for identifying messages in a historical repository;
- a time stamp containing the time the message was introduced in the MOM system;
- a correlation identifier that can be used to link the current message to a previous one, or to link it to an external information source;
- a reply field specifying the destination where a reply to the current message should be sent;
- a redelivery flag (used only when receiving messages) to indicate that the current message has already been delivered before;

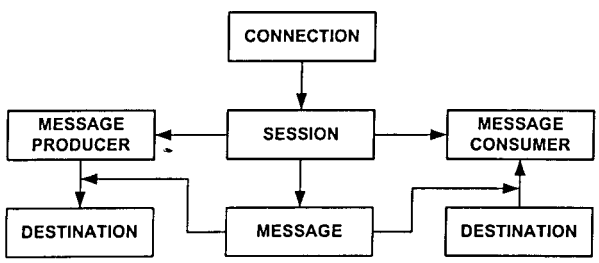


Figure 5.4: Behavior of JMS clients

- a type field that specifies a message type identifier (string value) supplied by the client when the message is being sent;
- an expiration value indicating that the message is only relevant within a certain time period, and that it should be discarded after that time;
- and a priority value (between 0 and 9) to make the MOM system deliver messages of higher priority ahead of messages with lower.

Besides these standard header fields, a message may include a set of properties, to be interpreted as optional header fields. Properties may contain application-specific values or parameters required by the particular MOM system being used. Properties are essentially name-value pairs that are set prior to sending the message and when a client receives the message, property values cannot be changed. Property may have several numeric and alphanumeric types such as byte, integer and string, and these types may be set at run-time. JMS reserves a set of pre-defined properties to help the MOM system with message exchange. These properties include information such as the identity of the user and application sending the message, the number of delivery attempts, and the time when the MOM system delivered the message to the consumer. Some properties are set when the message is sent, so they will be available to both producer and consumer. Others are set by the MOM system, and only the consumer will be able to read them. Furthermore, JMS reserves a name prefix for system-specific properties that may be required by some MOM products.

JMS realizes that applications often need to sort and filter messages they exchange with other applications. Especially when a message is sent to more than one client, it is useful to include selection criteria in the message header or as part of the message properties. This way the MOM system can handle the filtering and routing work, and avoid sending messages to clients that would discard them anyway. The message selection criteria are expressed according to *selector expressions*, which allow clients to specify the messages they are interested in, based on message header and properties. Selector expressions cannot reference message body values, since the message body is application-specific. The selector expressions use a syntax based on a subset of SQL, where query parameters are header and property values.

The message body contains the application data being sent, and it may have one of five different forms. It may be a stream message, whose body contains a stream of primitive values to be filled and read sequentially, or it may be a map message that contains an arbitrary number of name-value pairs. Both the stream and map message bodies support the same set of primitive data types. A message body may also be a text message, or a byte message with a stream of uninterpreted bytes, or even a message that contains a serialized Java object.

5.1.1.2 The messaging facilities

JMS specifies a general message interface containing attributes that are common to every message, and it provides specialized interfaces for each type of message. The general message interface deals with header fields and properties, while the specialized interfaces include those operations that are particular to each type of message body. This inheritance relationship is illustrated in figure 5.5, together with other main JMS interfaces. From this diagram it can also be seen that the distinction between point-to-point and publish/subscribe messaging is expressed as a specialization of the connection, session, destination, message producer, and message consumer interfaces. For each of these interfaces there are two specialized interfaces: one dealing with queues and another with topics.

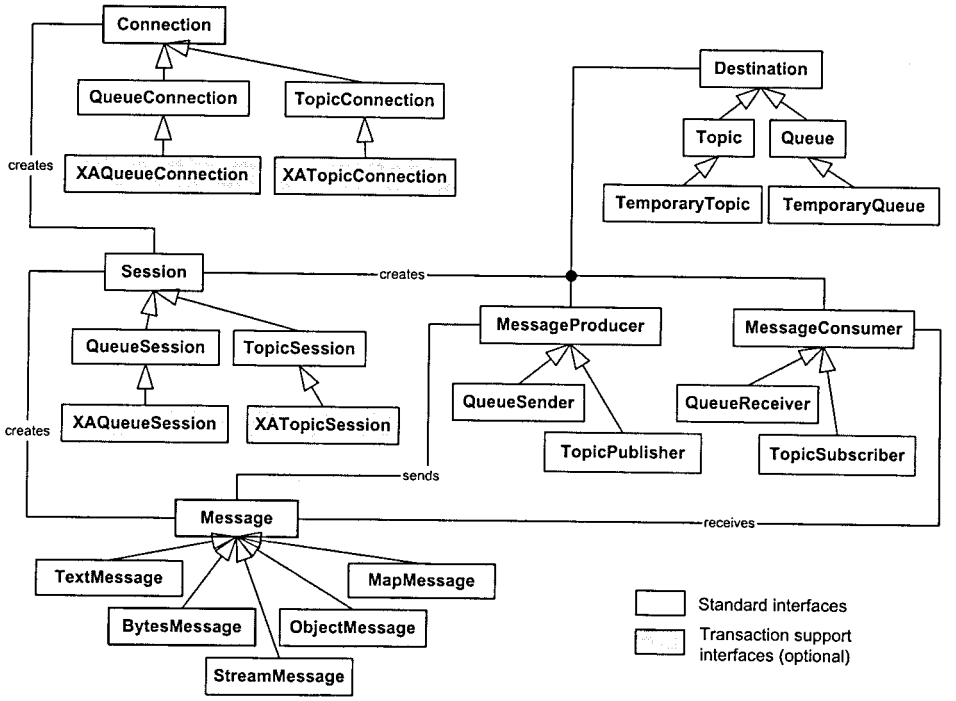


Figure 5.5: Simplified diagram of JMS interfaces

The Connection interface represents an active connection between the JMS client and the MOM system, and it is expected that JMS clients will use a single connection for all their messaging needs. When creating a Connection object, the client supplies a name and password, and these credentials will be used for authentication. After creation, the connection is not immediately active, and no messages are delivered to this connection until it has been started. This is achieved by invoking a specific start() method, which indicates that the client is ready to receive messages. Otherwise, clients would have to be able to handle incoming messages while they are connecting to the MOM system. By invoking the start() and stop() methods on a Connection object, clients are able to interrupt the delivery of messages, though they are still able to send messages. A close method is used to close the connection to the MOM system.

The Connection is a factory for Session objects, which are single-threaded contexts for producing and consuming messages. On its turn, the Session is a factory for other objects, namely Destination, MessageProducer, and MessageConsumer. Regarding the destination of messages, JMS does not define an address syntax; instead, the Destination interface encapsulates system-specific addresses, assuming that these addresses can be expressed as a string value. Thus, both Queue and Topic objects have names that depend on the particular MOM system. These names are used as the first header field in message being sent. It is also possible to create temporary destinations (TemporaryQueue and TemporaryTopic objects) which are convenient for example as reply destinations. The lifetime of a temporary destination is that of the connection, and any session on a given connection may create message consumers for a temporary destination.

There are some issues about sessions worth noting. One is that since a client may create multiple sessions, each session is an independent producer or consumer of messages. For example, if two

sessions have a subscriber for the same topic, both subscribers independently receive a copy of the original message. Another issue is the order of messages: JMS requires messages to be delivered in the same order as they were sent. However, when the MOM system dispatches messages, messages of higher priority may jump ahead of previous, lower priority messages. Message order is considered separately within each delivery mode: for example, a later non-persistent message may arrive ahead of an earlier persistent message, but it cannot arrive ahead of an earlier non-persistent message with the same priority. A third issues concerns message acknowledgment: JMS requires clients to acknowledge the receipt of messages from the MOM system. This can be done automatically by the Session object whenever a message is received, or it can be done by the client itself. A message that remains without acknowledgment will be delivered again to the message consumer.

A message consumer can be created with a message selector in order to restrict incoming messages to those which match the selection criteria. Message consumers allow clients to receive messages in synchronous or asynchronous mode. In synchronous mode the client explicitly requests messages from the message consumer. In asynchronous mode the client provides an object that implements a MessageListener interface, which has a single method to be called by the MOM system whenever a message arrives. Message acknowledgment is only required in asynchronous mode.

As shown in figure 5.5, JMS includes a set of optional interfaces that encapsulate transactional behavior. In fact, JMS does not require a MOM system to support transactions, but if it supplies this functionality then it should implement those optional interfaces. These interfaces are based on another API, the Java Transaction API (JTA) [Sun, 1999]. A MOM system that implements these interfaces is able to participate in a distributed transaction processing system that uses a two-phase commit protocol. The two crucial interfaces are the XAQueueSession and the XATopicSession, which can be referred to as *transacted sessions*. To create transacted sessions, XAQueueConnection and XATopicConnection must be used, respectively. A transacted session supports a single series of transactions, where each transaction groups a set of messages in an atomic unit of work. Usually, a transaction will comprise a set of messages to be sent, or a set of messages that must be received. The transaction is committed if all messages have been successfully sent or if all messages have been received. In this last case, message acknowledgment is handled automatically when the transaction is committed.

It is possible to include outgoing and incoming messages in the same transaction, but special care must be taken:

- Incoming messages must not depend on outgoing messages, such as one message being a reply to a previous one. If both a request message and its reply are included in the same transaction the JMS client will block: the sending operation cannot take place until the transaction is committed, and the transaction will never be committed because, without sending the request, no reply will be received.

- The transaction context does not flow with the message. For example, the production and receipt of a message cannot be part of the same transaction because transactions take place between a client and the MOM system, not between clients. Production and consumption of messages in a session can be transactional, but not for a specific message across sessions. This is due to the fact that MOM systems decouple message producers and message consumers. Thus, reliability cannot be achieved by imposing transactions across applications, it must be provided by the MOM system itself.

5.1.2 The CORBA Notification Service (CosNotify)

The CORBA Notification Service (CosNotify) [OMG, 2000b] is, like JMS, a specification of the interfaces that a MOM system should provide. But whereas JMS is to be implemented in Java, CosNotify, which builds on CORBA, is both platform- and language-independent. The CORBA Notification Service is a comprehensive specification, perhaps even too extensive to be implemented simply as an interface for an existing MOM system. CosNotify requires several features, such as event filtering and Quality of Service (QoS) characteristics, so it is difficult for a MOM system to implement these features if it does not already support them in the first place. For this reason, it is not common to find MOM systems that implement the CORBA Notification Service specification. Instead, it is easier to find systems that have been specifically built as implementations of CosNotify, as shown in figure 5.6. Except for Tuxedo, which is a commercial application integration system, all other products have been purposely developed as implementations of the CORBA Notification Service.

5.1.2.1 The event channel

The CORBA Notification Service is based on the concept of *events*. In practice, an event is essentially a reference to a message object that CosNotify conveys from one application to other applications. An application is a supplier if it produces events, and a consumer if it receives events. These are equivalent, respectively, to message producers and message consumers in JMS. Both suppliers and consumers connect to a common event channel, which is a central component of the CORBA Notification Service. The event channel receives events from any supplier and forwards them to every consumer connected to that channel. CosNotify requires this event channel to comply with the CORBA Event Service (CosEvent) [OMG, 2001] specification. In fact, the CORBA Event Service was the basis for developing the CORBA Notification Service.

The CORBA Event Service specifies two modes for event delivery: one is called the push model and other is the pull model. In both cases, events flow from suppliers to consumers. With the push model, suppliers push events to the event channel, and the event channel pushes events to consumers. With the pull model, the event channel pulls events from suppliers, and consumers pull events from the event channel. In client/server terminology, this corresponds to an interchange of roles between the event channel and each application: for example, when the event channel pushes a message to a consumer, the channel is the client and the consumer is the server, but when the consumer pulls a message from the event channel it is the other way around. In figure 5.7, the server is always on the side the arrow points to, with the client being on the origin of the arrow.

Suppliers and consumers are free to operate according to either the push or pull model. This means that they may operate according to the same model, as shown in figure 5.7, or according to

Vendor	URL	Product	Open source
AT&T Research	http://www.research.att.com/	omniNotify	x
BEA Systems	http://www.bea.com/	Tuxedo	
DSTC	http://www.dstc.edu.au/	dCon	
IONA	http://www.iona.com/	ORBacus Notify	x
OCI	http://www.theaceorb.com/	TAO Notification Service	x
PrismTech	http://www.prismtechnologies.com/	OpenFusion Notification Service	

Figure 5.6: Some providers of CosNotify implementations (August 2002)

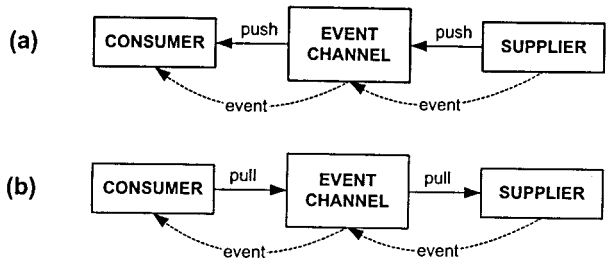


Figure 5.7: Push (a) and pull (b) models in the CORBA Event Service

different models. Figure 5.8 illustrates the four possibilities. The communication of events between supplier 1 and consumer 1, and between supplier 2 and consumer 2, is the same as depicted in cases (a) and (b) of figure 5.7, respectively. However, communication between supplier 1 and consumer 2, and supplier 2 and consumer 1, leads to hybrid push/pull and pull/push models. In the first case, between supplier 1 and consumer 2, the event channel works much like a regular message queuing system, because it merely stores events from suppliers until they are pulled by consumers. In the second case, between supplier 2 and consumer 1, it is the responsibility of the event channel to pull events from suppliers and push them to consumers. In this case, because it is the event channel which initiates the flow of events between suppliers and consumers, it has been said that the event channel plays an equivalent role to that of an intelligent agent [Henning and Vinoski, 1999].

The CORBA Event Service specifies a set of interfaces that allow suppliers and consumers to communicate, using push and pull models, through an event channel. This set of interfaces concerns mainly the interfaces between consumers and the event channel, and between suppliers and the event channel. For the consumer, the event channel appears to be a supplier, while for the supplier it appears to be a consumer. The interfaces that make the event channel resemble a supplier to consumers, and a consumer to suppliers, are called the *proxy supplier interface* and the *proxy consumer interface*, respectively. These proxy interfaces do not correspond to particular suppliers or consumers, they are just entry points to the event channel. Suppliers use the proxy consumer interface to submit events to the channel, while consumers use the proxy supplier interface to retrieve events from the channel, as illustrated in figure 5.9.

5.1.2.2 Typed and untyped events

The CORBA Event Service specifies events as being typed or untyped. Untyped events are declared as being of type any. This means that the channel is able to convey any object as an event. It also simplifies the CosEvent specification, and it ensures that event data can be passed unchanged through

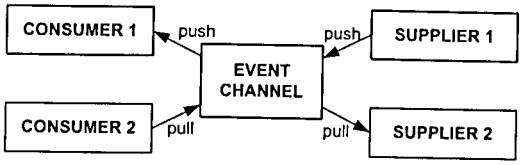


Figure 5.8: Suppliers and consumers operating in push and pull models



Figure 5.9: The proxy supplier and proxy consumer interfaces

any given implementation of the CORBA Event Service. But, on the other hand, this feature assumes that both suppliers and consumers must previously agree on the contents of an event. In fact, every consumer must be aware of these contents, since the produced events are delivered to all consumers connected to the channel. Of course, CORBA provides mechanisms to retrieve information about an object represented by a reference of type any, but those mechanisms are of little use if suppliers and consumers cannot semantically understand each other. In practice, applications must define the event structure prior to interaction and then, at run-time, they must package that data structure into an any object.

With typed event communication, instead of exchanging events, consumers and suppliers call operations on each other, according to a mutually agreed interface I. In the push model, suppliers call operations on consumers using interface I. This interface must obey two restrictions: all parameters must be input parameters, and no return values are allowed. Incidentally, these are the restrictions CORBA imposes on oneway operations. These restrictions can be explained by the fact that the Event Service decouples the supplier (which invokes the interface) from the consumer (which implements that interface). In fact, the supplier invokes only the proxy consumer, and it is the proxy supplier which will invoke the consumer. As a consequence, no return values or output parameters are allowed in the typed push model.

In the typed pull model, consumers call operations on suppliers using an interface that is the counterpart of I, denoted by Pull<I>. For every operation in I, interface Pull<I> contains two operations. One is the pull counterpart of the original operation with all input parameters changed to output parameters. The output parameters will carry the event data or, if no event is available, it will block. A second operation is basically the same but without blocking: if no event is available, the operation returns with its output parameters undefined. Therefore, Pull<I> is designed to allow pulling exactly the same event data that can be pushed with interface I.

Despite these possibilities, the CORBA Event Service has some limitations. The first limitation is that there is no event filtering: each consumer receives all events from all suppliers. In message queuing systems, however, events usually have a destination queue, and they are not to be received by other applications. Even if the data is not sensitive and can be sent to all consumers, this is somewhat wasteful since many of them will probably discard the event anyway. Another motivation for event filtering is that even with a single supplier, consumers may still receive events they are not interested in. A given consumer may be interested only in some kind of events from one supplier. A second main limitation is that CosEvent does not adequately support asynchronous messaging. For example, and depending on the particular implementation, a push supplier may block until the event is pushed to all consumers, or pull operations may block until the an event becomes available. A third issue is that there are several CORBA objects that pertain to the event channel, but CosEvent does not specify how to create these objects. In other words, the Event Service does not specify the factories for event channel objects. This allows vendors to implement these factories in their own way, which makes it difficult to use the same applications with different implementations of CosEvent.

5.1.2.3 Structured events

The purpose of the CORBA Event Service is to specify an event channel that decouples communication between applications. Using this event channel, the CORBA Notification Service specifies a MOM infrastructure supporting both message queuing and publish/subscribe messaging. The Notification Service extends the Event Service by introducing new capabilities, namely the ability to transmit events with well-defined structure, the ability to filter events, the ability to subscribe to event types, and the ability to configure QoS properties for channels and events. The Notification Service makes use of the same semantics as the Event Service, and in most cases its interfaces inherit from those of the Event Service, thus granting interoperability between CosEvent clients and CosNotify clients.

As shown in figure 5.10, the Notification Service supports all Event Service interfaces, including administration and proxy interfaces, plus several other new interfaces according to the types of event that CosNotify supports. Some of these interfaces are related to each other by inheritance relationships. For example, the CosNotify administration interfaces are specialized versions of the CosEvent ones. Thus, an application can connect to the Notification Service using one of three techniques: the first is to use the CosEvent administration interfaces to create CosEvent proxy objects; the second technique is to use the CosNotify administration interfaces to create the same CosEvent proxy objects; the third possibility is to use CosNotify interfaces to create CosNotify proxy objects exclusively. Furthermore, the Notification Service specifies a factory interface for event channels, so every implementation must comply with this interface in order to create these CORBA objects. Because the CosNotify architecture is hierarchical, all other objects will be subsequently created from some parent object.

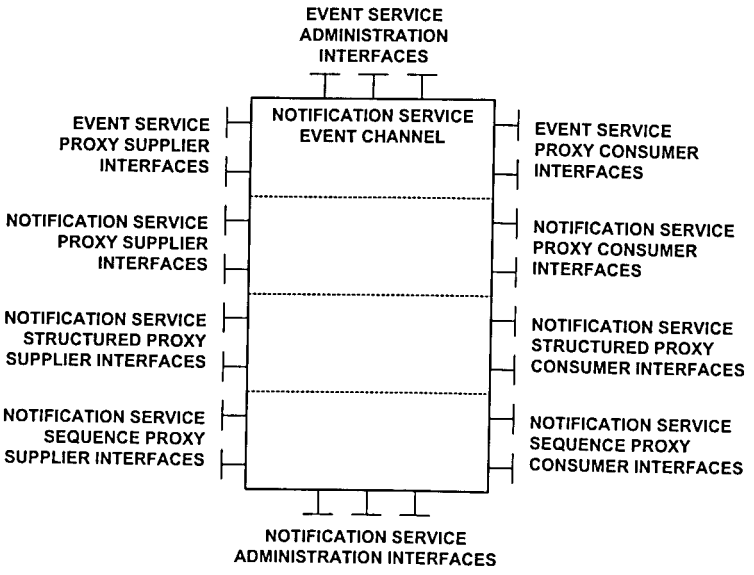


Figure 5.10: Interfaces specified by the CORBA Notification Service[70]

[70][OMG, 2000b]

Like the Event Service, the Notification service allows applications to exchange untyped and typed events. However, the Notification Service defines new proxy interfaces for applications that wish to send and receive *structured events*. The *structured proxy supplier interface* and the *structured proxy consumer interface* allow consumers and suppliers, respectively, to exchange structured events. The *sequence proxy supplier interface* and the *sequence proxy consumer interface* support sequences of structured events. Structured events provide a well-defined data structure that can be used in various event types. The new CosNotify proxy interfaces were specified so that structured events can be transmitted directly, without the need to package them into an any object, because the event structure is known both to the applications and to the Notification Service event channel. Structured events thus provide the advantages of typed event communication, without the difficulties found in implementing typed event communication as defined by the Event Service (interfaces I and Pull<I>).

A structured event has a general format comprising two main components: a header and a body. The header can be further separated into a fixed portion and a variable, optional portion. The fixed header portion, the optional header fields, and the event body can be regarded as corresponding directly to the header, properties and body in the JMS message model described earlier. The fixed header portion, as illustrated in figure 5.11, comprises three string fields. The domain_name concerns the industry domain where the event was defined, and the type_name classifies the type of event within that domain. The event_name uniquely identifies the specific instance of the event being issued. The combination of the domain_name and type_name can be used as an index in a event type repository, which contains a description of the fields for every event type. This event type repository is an optional feature of the Notification Service.

The variable portion of the event header contains an arbitrary number of optional header fields described by their name and respective value, the value being a reference to an object of type any. These fields may contain any application-specific data. As in JMS, CosNotify also reserves a predefined set of optional header fields, to be used in QoS-related information. The event body contains an additional set of similar fields, but this time to be used in filtering, hence the name of *filterable*

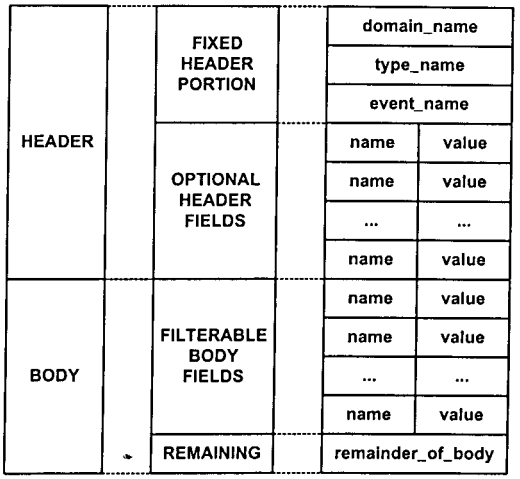


Figure 5.11: Structured event in the CORBA Notification Service[71]

[71][OMG, 2000b]

body elements. The remainder_of_body is an object of type any which, in a similar way to the Event Service, allows any data structure to be passed to the event channel.

5.1.2.4 QoS properties

One of the two main improvements the Notification Service offers when compared to the Event Service is that it specifies interfaces to control the Quality of Service (QoS) characteristics of event delivery. The Notification Service defines a number of QoS properties and their value types, since properties are pairs of string and any objects. QoS properties may be inserted as optional header fields in a structured event. In this case, the QoS properties apply only to that event, and new events will have to define their own QoS properties. In addition, the Notification Service allows QoS properties to be configured for individual proxy objects, for administration objects, or for a whole event channel.

By setting the QoS properties for a proxy object, every event sent or received through that proxy will have the same QoS configuration. On the other hand, if QoS properties are set for an administration object, then every proxy object created from that administration object will share the same QoS configuration. This is because CosNotify treats each administration object as the manager of the group of proxy objects it has created. If the QoS properties for the entire event channel are configured, then these will be applied to all events, proxy objects and administration objects, unless these objects define their own QoS properties. The Notification Service thus defines a four-level hierarchy with the ability to override QoS properties at each level.

The Notification Service defines several standard QoS properties:

- *Reliability* concerns event delivery and actually corresponds to two QoS properties, *event reliability* and *connection reliability*, both having two possible numeric values: *best effort* or *persistent*. If both are set to best effort then there will be no special delivery guarantees, and the same event may be delivered more than once. If connection reliability is set to persistent then the connections of consumers and suppliers to the event channel will be made persistent, and re-established upon system failure. If event reliability is set to persistent then all events will be stored persistently until they are delivered, or until they reach a given expiry limit.

- *Priority* is used to define the order for delivering events to a consumer. Priority is represented by a numeric value, where -32767 is the lowest priority and 32767 the highest, the default value being zero. However, it is still possible for a consumer to receive the messages in a different order if it uses filtering.

- The expiry time for an event can be specified by means of two properties called *stop time* and *timeout*. Stop time defines an absolute date and time after which the event can be discarded; timeout defines a time interval after which the event can be discarded.

- Earliest delivery time is used when an event should be held for some time before being delivered. *Start time* is a QoS property that defines an absolute point in time, after which the event can be delivered.

- *Maximum events per consumer* is used to limit the maximum number of events that the event channel should hold before delivery to a consumer. When this limit is reached the channel will prevent further events to be queued for that consumer. If this property is set for the whole event channel, then it will apply to all consumers connected to that channel. Its default value is zero,

which is equivalent to having no limit on the number of queued messages. This property has no meaning for suppliers and proxy consumers.

- *Order policy* is used to order the events before delivery, and it must have one of four constant values: *any order* means that no special order is desired, *FIFO order* means that events should be delivered according to their arrival, *priority order* indicates that higher priority events should be delivered ahead of lower priority ones, and *deadline order* specifies that events with the shortest expiry time should be delivered first.

- *Discard policy* is employed whenever the number of messages in a queue exceeds the maximum queue size. It has five values, four of which are equivalent to those of the previous property. For example, *FIFO order* means that the first event received should be the first to be discarded. A fifth possibility is the *LIFO order*, specifying that the last event received should be the one to be discarded first.

- *Maximum batch size* refers to the maximum number of events that will be delivered within each sequence of structured events. This QoS property applies only to sequence proxy suppliers and sequence proxy consumers, and its default value is one.

- *Pacing interval* also concerns sequences of events. It defines the maximum period of time the channel will collect events before delivering them as an event sequence to the consumer. If the number of events received within the pacing interval exceeds the maximum batch size then the consumer will receive a sequence with length equal to that value. On the other hand, if no events arrive during the pacing interval, the proxy supplier will wait for at least one event to arrive. The default value for the pacing interval is zero, meaning that no time restriction will be considered and that the sequence will be delivered as soon as it contains a number of events given by maximum batch size.

Some of these QoS properties may be set at all levels, while others are restricted to some levels. For example, the start time and stop time properties can only be used on a per-event basis, while the order policy property cannot be used at this level. Figure 5.12 shows the levels that each QoS property can be applied to.

Property	Channel	Admin.	Proxy	Event
Event reliability	x			x
Connection reliability	x	x	x	
Priority	x	x	x	x
Start time				x
Stop time				x
Timeout	x	x	x	x
Max. events per consumer	x	x	x	
Order policy	x	x	x	
Discard policy	x	x	x	
Max. batch size	x	x	x	
Pacing interval	x	x	x	

Figure 5.12: Applicability of QoS properties at various levels[72]

[72][OMG, 2000b]

In general, QoS properties should be consistent throughout the whole path an event travels. For example, a supplier may have QoS properties set to persistent event reliability while, at the opposite end, the consumer may set the QoS for its proxy supplier to best-effort delivery. Thus, it becomes impossible to guarantee a QoS requirement without the agreement of both parties. Moreover, the event channel must also apply the same QoS properties. The Notification Service realizes that most of the responsibility of ensuring end-to-end QoS properties belongs to suppliers and consumers, so it leaves this issue to be handled by external applications. However, the Notification Service does specify some mechanisms to detect conflicting QoS specifications, and to support applications in setting and validating common QoS properties.

5.1.2.5 Event filtering

The other major improvement the Notification Service introduces is the possibility of filtering events. The default behavior of proxy objects is to forward all events they receive, but CosNotify allows different behavior by specifying two kinds of filters to be associated with proxy objects. The first kind are *forwarding filters*, which can be attached to every type of proxy object to constrain the events that will pass through that proxy. The second kind, *mapping filters*, can only be attached to supplier proxy objects. They allow, for example, a consumer to change the priority or expiry time of any received event. In general, mapping filters affect the way a proxy treats events with respect to these two QoS properties.

Both kinds of filters, forwarding filters and mapping filters, can be associated either explicitly or implicitly with a proxy object. When filters are associated with a specific proxy instance the association is explicit. But it is also possible to associate filter with an administration object that creates proxy objects. This is because every proxy created from an administration object will have, implicitly, the same filters as that administration object.

Forwarding filters affect the event forwarding decisions of proxy objects. A forwarding filter represents a constraint that is evaluated for every event. If the event obeys the constraint, it will be forwarded; otherwise, the event will be discarded. The constraint of a forwarding filter is composed of two elements. The first element is a sequence of the types of events the constraint applies to. This includes, for each type of event, its domain_name and type_name as depicted in figure 5.11. The second element of the constraint contains a boolean expression based on filterable body fields, to be evaluated for each particular event. This boolean expression follows a filter constraint language, which is the Extended Trader Constraint Language [OMG, 2000b].

When a proxy receives an event, it invokes its associated filter object. If the filter applies to that kind of event, the event is matched against the filter constraint. If the filter contains more than one constraint, the event will be forwarded if it matches at least one constraint, otherwise it will be discarded. Another possibility is for a proxy to have several filters attached to it. In this case, the event will be forwarded if it is accepted by at least one filter. Event filtering is applied immediately upon receipt, so an event that is to be discarded never reaches the queue of events to be delivered by the proxy. On the other hand, the fact that an event is accepted for forwarding does not mean the event will be delivered immediately; it may have to wait on the queue behind other events.

Mapping filters work exactly the same way, except for the fact that, besides evaluating a constraint, they return a new value to be assigned to a QoS property of that event. The Notification Service realizes that mapping filters are especially useful to change event priority and expiry time, but they are sufficiently generic to be applied to other QoS properties as well. A different mapping filter must be attached to a proxy object in order to deal with each QoS property. Besides identifying the type

of event and defining boolean constraints, the mapping filter contains two additional elements: one is the new value for the corresponding QoS property, and the other is a default value for that same property.

A proxy having a mapping filter for the priority property, for example, will invoke that filter as soon as a new event arrives. The filter will successively evaluate its constraints for this new event until either the event satisfies one constraint or it is found to satisfy none. As soon as the mapping filter finds a constraint the event obeys to, it will return the new value for the priority property. The proxy object will then assign the new priority value to the event. The same procedure may apply, with another mapping filter, to the expiry time property.

If, however, the event is found to satisfy no constraint, the mapping filter will return a default value for the QoS property. Resuming the priority example, this value will be used as the new priority value if the proxy object cannot find another priority value for the event. First, the proxy will consider the priority value to be the one defined as a header field in the event. But if this optional field is missing, the event will be assigned the same priority value as that defined in the current proxy, or in the previous proxy which has forwarded the event. Still, if the property value is undefined on these proxy objects, then the proxy will assign the event the default priority value given by the mapping filter.

5.1.2.6 Event translation

Internally, the Notification Service uses structured events as depicted in figure 5.11. But because the Notification Service is backward compatible with the Event Service, it must allow CosEvent applications to use the same event channel. Since these applications may use typed and untyped events as defined by the Event Service, the Notification Service specifies how to translate events of one kind to another. In fact, the Notification Service supports the translation of message format in every situation where the supplied event is of a different format from the one a given consumer was designed to receive. Thus, there are six possible translations:

- translation from any to structured event - the original event is packaged into the remainder_of_body on a structured event, shown in the bottom of figure 5.11, which is of type any;
- translation from typed event to structured event - a new structured event object is created and its filterable body fields are populated with the contents of the typed event: the first of those fields (called operation) will contain the name of the invoked operation on interface I, and the remaining fields will contain the parameter name and values passed during invocation by the supplier;
- translation from structured event to any - a new any object is created and the structured event is packaged into this object;
- translation from typed event to any - a new any object is created containing a sequence of name-value pairs: the first of these will contain the name of the invoked operation on interface I, and the remaining elements will contain the parameter name and values passed during invocation by the supplier;
- translation from any to typed event - in this case, translation is possible only if the original event contains a sequence of name-value pairs, whose first element is the name of the operation

on interface I, and the remaining elements correspond to the respective parameters to be used during invocation;

- translation from structured event to typed event - again, this translation requires the structured event to contain a sequence of name-value pairs, which correspond to the operation name and parameters to use when invoking interface I.

This translation scheme is not entirely reversible since, for example, the translation from any to structured event and then back from structured event to any does not result in the original any object. To avoid this situation, the Notification Service defines a special event type denoted by %ANY. This value is to be used as the type_name field in figure 5.11 whenever the structured event contains an any as the result of the translation from any into structured event. Every proxy consumer that accepts structured events, or sequences of structured events, must check the type of an incoming event. Whenever such type is %ANY, the proxy must extract the original any object from the remainder_of_body on the structured event, before performing further translations. Similarly, when receiving an any event, the proxy should check whether it contains a structured event or not. If that is the case, the structured event should be extracted from the any object before performing further translations.

5.2 Business-to-business frameworks

In general, MOM systems focus on the underlying mechanisms that enable message exchange between applications. Typically, those systems implement message queues and control message routing and delivery taking into account parameters such as expiry dates, priority and persistence. The purpose of MOM systems is to serve as a common integration infrastructure between enterprise applications, and to support the exchange of any data between those applications. As a result, MOM systems provide general messaging functionality regardless of any message format in particular. Except for some attributes that the MOM system inserts for delivery and QoS purposes, the message format is absolutely application-specific.

Business-to-business (B2B) frameworks take a seemingly opposite approach: in many cases they define a set of common messages and their format, regardless of the underlying exchange mechanisms. The purpose of B2B frameworks is to enable business-to-business interoperability by specifying a set of common business documents. The development of B2B frameworks is based on the principle that, in spite of a globally connected and accessible network, enterprises still have to agree on a common business language before they can perform commercial exchanges through the Internet, i.e., before they can achieve B2B interoperability.

The first step towards this goal has already been taken in the past with EDI. But the increasing use of Web protocols, such as HTTP, and the remarkable success of HTML have favored more flexible solutions, notably XML. Hence, most B2B frameworks build heavily on XML. Typically, a B2B framework is an XML-based, middleware-neutral document specification, though usually most B2B frameworks require the use of Internet and Web protocols, such as HTTP, SSL, and MIME.

Originally, B2B frameworks focused solely on developing vendor-independent specifications for a set of documents to be exchanged between business partners. These specifications were usually developed with a particular industry or service sector in mind. For example, the Data Interchange Standards Association has several affiliated organizations[73] that have developed standards for particular sectors, namely MISMO for the mortgage real estate, CIDX for the chemical industry, mpXML

[73] http://www.disa.org/affiliates.cfm

for the meat and poultry industry, HEDNA for hotels, OTA for the travel industry, and the IFX Forum for financial services. In fact, financial services is one of the areas where B2B frameworks have found many applications: the FIX Protocol[74], the FpML language[75], and the OFX specification[76] are all B2B frameworks that define a set of common XML documents to be used in financial exchanges between organizations.

Lately, other frameworks have realized the need to coordinate the actions of different business partners. Business partners should definitely agree on the structure of documents they exchange, but they should also know when to exchange those documents and how to articulate those external exchanges with their internal business processes. As a result, B2B frameworks have begun focusing both on document format and on inter-enterprise business processes that concern the exchange of those documents. From these B2B frameworks, some frameworks specify in detail the sequence of message exchanges between partners; other frameworks specify the infrastructure required for business partners to implement those exchanges. In other words, some frameworks specify both message format and exchange sequence, and some specify the message format and an infrastructure which allows business partners to define and implement their own interactions.

5.2.1 OBI

The purpose of the Open Buying on the Internet (OBI) specification [OBI, 2001] is to develop a common framework to guide companies in implementing interoperable, Internet-based purchasing systems. OBI focuses on high-volume, low-value transactions that are believed to account for about 80% of most organizations' purchasing activities. The OBI framework is based on a particular model of business-to-business interaction, which entails a *buying organization*, a *requisitioner* belonging to that buying organization, a *selling organization*, and a *payment authority*, as shown in figure 5.13.

5.2.1.1 The OBI interaction model

According to the OBI framework, the requisitioner uses a Web browser to retrieve product information from the selling organization. The selling organization authenticates the requisitioner based on information presented in the requisitioner's digital certificate, before giving access to its online catalog (step 1). The requisitioner can then browse the catalog and select the desired items. Once the

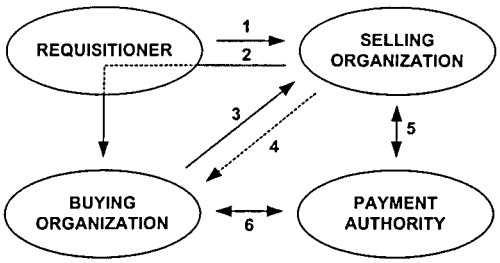


Figure 5.13: The OBI business-to-business interaction model

[74] http://www.fixprotocol.org/
[75] http://www.fpml.org/
[76] http://www.ofx.net/

requisitioner has chosen the items, the selling organization sends an order request to the buying organization, either directly or via the requisitioner (step 2). At the buying organization, the order request may have to go through an internal process for approval, or may otherwise have to be completed with administrative information.

The completed and approved order request is then converted into an digitally signed order to be transmitted to the selling organization (step 3). The selling organization may optionally return an order response document or a ship notice with the details of order shipment (step 4). After the selling organization obtains credit authorization from a payment authority (step 5), the order enters the supplier's fulfillment process and is delivered to the requisitioner. The payment authority then issues an invoice and receives payment from the buying organization (step 6).

5.2.1.2 OBI documents and data objects

The order request and the order are two fundamental documents, and the OBI framework specifies these documents in detail. For this purpose, OBI relies on EDI standards, so these documents are specified according to an EDI syntax. More recently, OBI has introduced XML-based specifications for the same documents. Currently, OBI allows documents to be expressed in either one of these formats. Figure 5.14 shows the similarities between both formats. In figure 5.14(a) many of the segments shown are due to the structure of EDI documents, and figure 5.14(b) shows the structure for XML documents. Apart from the language-specific structure, the two formats are equivalent.

Regardless of the particular format being used, OBI requires documents to be encapsulated in *data objects* before being transmitted. An OBI data object is a linear data structure comprising three main fields: version number, OBI data, and signature. The version field determines the OBI version being used, and it specifies whether the document being transmitted follows an EDI or XML format,

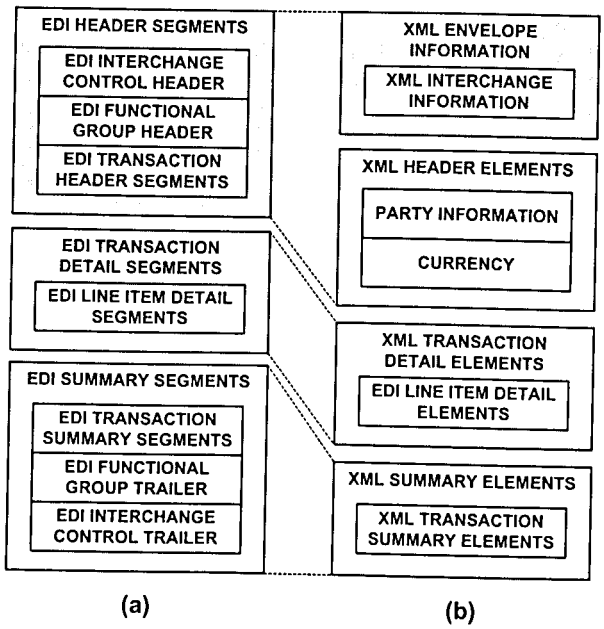


Figure 5.14: EDI (a) and XML (b) OBI data structures

as depicted in figures 5.14(a) and 5.14(b). The document goes into the second field, the OBI data field. The signature field is optional, but if it is present it should contain a digital signature of the document, i.e., a signature on the contents of the OBI data field.

5.2.1.3 Security measures

According to OBI, each entity shown in figure 5.13 must have its own digital certificate for authentication purposes. For example, the selling organization authenticates the requisitioner before giving access to its online catalog, and the buying organization authenticates the selling organization before transmitting an order document. The OBI authentication model is based on the use of digital certificates and the SSL protocol to ensure that, in every exchange, both sender and receiver have a cryptographic assurance of the identity of each other. In addition, these digital certificates are used to generate the digital signatures of documents to be included in the data object. The data objects are subsequently encoded in base64 [IETF, 1996] before being transmitted through secure HTTP connections.

5.2.2 cXML

The Commerce XML (cXML) framework [Ariba, 2001], like OBI, focuses on procurement processes, but it supports the exchange of documents between buyers, suppliers, and intermediaries such as e-marketplaces. The cXML framework specifies three main types of documents: the Catalog, the PunchOut, and the Purchase Order. The Catalog is a document that describes the products or services offered by a supplier, together with their respective prices. The cXML framework assumes that there are procurement applications at the buying organization that can read and store locally the information contained in Catalogs. Afterwards, local users from the buying organization will be able to browse the catalog information and to create Purchase Orders for particular items.

5.2.2.1 The PunchOut concept

The central feature of the cXML framework is the specification of the PunchOut. In reality, a PunchOut is more than a document, it is a protocol for interactive sessions between buyer and supplier. The PunchOut is an alternative to static catalog information: it enables remote communication between procurement applications at the buying organization and a special-purpose PunchOut site at the supplier. The cXML framework specifies how to implement a PunchOut site, which is basically a Web site that allows procurement applications at the buying organization to browse product information remotely. From their desktop, users are able to browse product options, specify configurations, and select the delivery method. Once these items have been chosen, the PunchOut site returns this information to the procurement application at the buying organization, which will now be able to create the corresponding Purchase Orders.

An additional possibility defined by cXML is to have PunchOut chaining, i.e., the buying organization establishes a PunchOut session with an e-marketplace and the e-marketplace establishes a PunchOut session with the supplier. This scenario can be extended to an arbitrary number of intermediaries between buyer and supplier. By having intermediaries such as e-marketplaces, a buyer can interact, using PunchOut, with several suppliers at once, placing orders for different products at different suppliers simultaneously. The cXML framework specifies that messages must contain path elements so that they can be routed to the intended supplier.

Each PunchOut session between a procurement application and a PunchOut site is composed of four different steps, as shown in figure 5.15. First, the user invokes the procurement application in order to search local catalogs for the desired product. If product information is available via PunchOut rather than by a local Catalog, the procurement application logs into an external hub or e-marketplace that will play the role of *PunchOut dispatcher*. The hub authenticates the buying organization and opens a secure session with the supplier. The cXML representative (Ariba) is a vendor of products that implement this functionality.

The document used during this first step is the PunchOutSetupRequest. As with all documents within a PunchOut session, this document travels through the Internet using secure HTTP connections. The supplier will return a PunchOutSetupResponse containing a URL that the procurement application will use to initiate a browsing session at the supplier's site. The second step is to use this session to select the desired products. During this step, the user experience is similar to that of browsing a regular Web site. The third step is to check out: the contents of the shopping cart are sent, through the hub, to the procurement application at the buying organization. The application receives a PunchOutOrderMessage containing product details and prices. Business processes that are internal to the buying organization take over from this point. For example, orders may have to undergo an approval process before being submitted to suppliers. As soon as the order is approved, the fourth and final step is to convert the order to a set of PurchaseOrder documents and send them, via the hub, to the suppliers.

5.2.2.2 cXML messages

The cXML framework supports two interaction models: request/response and one-way messages. For example, step 1 in figure 5.15 follows the request/response model, while steps 3 and 4 are one-way messages. The message format is slightly different for each of these interaction models. In the first model, the request message envelope has a header portion and a request portion. The header portion has four fields, as shown in figure 5.16(a). The From and To fields contain the identification of the originator and destination for the request, respectively. The Sender field identifies the user that opened the HTTP connection in order to transmit the message, and it is used for authentication purposes. The UserAgent field identifies the product name for the hub that is being used.

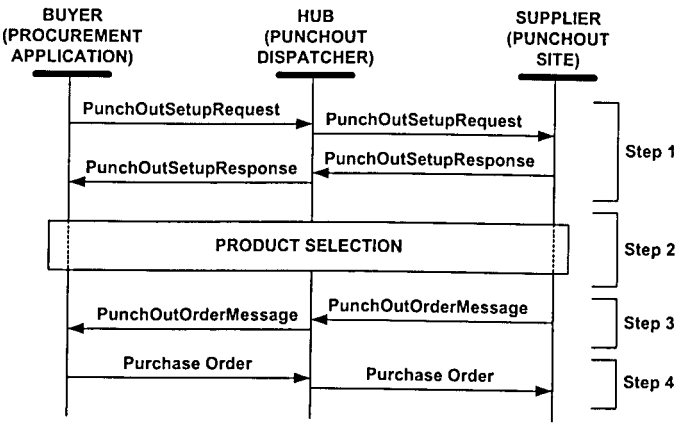


Figure 5.15: Message sequence for a PunchOut session

The request portion, which is equivalent to the message body, may contain any application-specific data, including file attachments. To support this, cXML requires messages to be encoded with MIME before being sent through HTTP. The same happens with the response message which, despite having no header portion, may carry any application-specific data, as shown in figure 5.16(b). The one-way message, depicted in figure 5.16(c), is similar to the request message except for the request portion, which is replaced by an equivalent message portion. All messages have an envelope with a unique message identifier and a time stamp.

5.2.2.3 Master agreements

Another feature of cXML is that it specifies the possibility of establishing agreements, called *master agreements*, between buyers and suppliers. A master agreement establishes the commitment of a supplier to provide a certain amount of products according to an fixed price within a certain period of time. The cXML framework specifies the MasterAgreementRequest document to be submitted by the buying organization to the supplier. This document has a set of elements that define the agreement date and its expiration date, the maximum and minimum price values, and the quantities for each product.

5.2.3 BizTalk

The BizTalk framework [Microsoft, 2000], like cXML, specifies the message format that encloses documents to be exchanged between applications at different sites. At each site, BizTalk assumes that there is a server which is capable of producing and consuming messages according to the specified format. In addition, BizTalk suggests the use of particular protocols for communication between these servers. This is referred to as transport bindings, i.e., the exchange of BizTalk messages using certain transport protocols. However, the BizTalk framework is independent of the implementation details of those protocols.

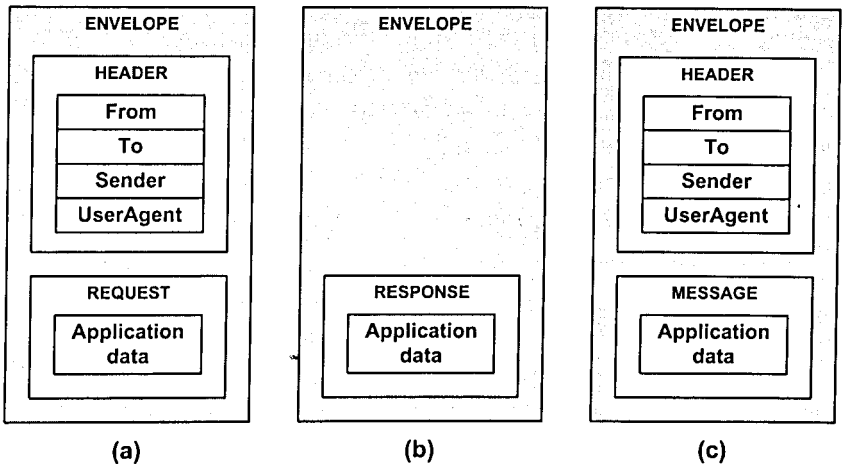


Figure 5.16: (a) Request, (b) response, and (c) one-way cXML messages

5.2.3.1 Framework layers

According to BizTalk, each site or node, within the same enterprise or belonging to different enterprises, is represented as an abstraction of three layers comprising the applications, the BizTalk-compliant server, and the transport protocols, as shown in figure 5.17. At each node, the applications produce and consume business documents. When applications produce documents, they submit them to the BizTalk-compliant server, to be sent to another node. BizTalk does not require these documents to comply with the specified format, though it is expected that the documents are expressed in an XML format. The interface between the application and the BizTalk server is not addressed by the BizTalk specification. In general, it is the job of the BizTalk server to create a BizTalk-compliant message that is suitable for being sent with the transport protocols being used. For example, the BizTalk server uses specific XML tags to identify the transport-specific destination address.

5.2.3.2 Message format

The BizTalk message format is composed of a header portion and a body portion, both of which are inserted into an *envelope*, as shown in figure 5.18. The purpose of the envelope is to define the XML namespaces [W3C, 1999a] used within the message. The header portion is expressed according to a syntax and semantics that are fully specified by the BizTalk framework. On the other hand, the body portion carries XML documents that may use other XML namespaces, and whose structure is to be defined by applications. Originally, the BizTalk framework included a library of common document formats that applications could use. This library was maintained by an online community, and was organized according to several industry sectors. Later, however, BizTalk has changed its focus to the functionality of BizTalk server. Nowadays, BizTalk is more of a commercial product than a B2B framework, but some points are still worth noting.

The header portion has several entries: endpoints and properties are mandatory entries, while services, manifest and process are optional. The endpoints header entry is used to identify the message origin and destination as business-related names or transport-specific destination addresses, depending on the implementation. The properties header entry identifies the message in a unique way, it records the time when the message was sent, it specifies an expiry date for the message, and it refers, via a URL, to an external document specification that is used to verify the consistency between the purpose and contents of the message. The services header entry is used to request a reply from the destination BizTalk server, when the message is received or when the message is processed at the receiving end. The manifest contains a set of references to other business documents or file

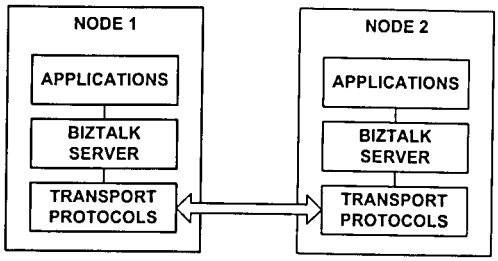


Figure 5.17: The BizTalk framework layers

attachments that are considered to be part of the message, regardless of whether they are physically included in the BizTalk message or not.

The process header entry provides the ability to relate a BizTalk message with a business process running on either the sending or the receiving node. Alternatively, it may also refer to a commonly agreed business process in which both nodes participate. The process header entry contains three tags: type, instance, and detail. The type tag contains a reference to the type of business process and it is equivalent to the name of a process definition. The instance tag identifies the specific process instance being run. It may be expressed, for example, as an extension to the previous tag, containing both the process name and a sequence number that identifies the process instance. The detail tag gives further information that may be required, typically, to identify an individual step or activity within that process instance. Its contents are dependent on the implementation and on the applications being used.

5.2.3.3 Technology bindings

BizTalk uses SOAP [W3C, 2000b] as the starting point for defining messages. As a consequence, BizTalk messages contain several elements that are particular to SOAP. For example, both the header and body portions are enclosed in SOAP-specific tags. Because SOAP is usually employed in combination with HTTP, it is not surprising that BizTalk specifies transport binding using HTTP. BizTalk messages may include documents with attachments, so BizTalk also defines how to enclose a BizTalk message in a multipart MIME message [IETF, 1996]. To enable encryption and digital signatures, BizTalk requires the use of S/MIME [IETF, 1999]. And because MIME is typically used in e-mail, it should be straightforward to use e-mail protocols as transport protocols for BizTalk messages.

5.2.4 bolero.net

Whereas the BizTalk framework can be implemented by acquiring and installing a BizTalk-compliant server for each site, bolero.net [Bolero, 1999a] is a running system owned by an independent third-party. Created as a joint effort of a transport insurer (TT Club) and a financial community (SWIFT),

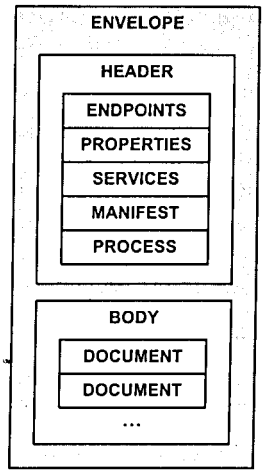


Figure 5.18: The BizTalk message format

bolero.net allows trading parties to exchange contractual and fulfillment data online in a secure environment. The bolero.net system was designed to support trading processes and information exchange between importers, exporters, freight forwarders, port authorities, inspection agencies, carriers, ship agents, customs agencies, and financial institutions. The system has two main components: the Core Messaging Platform, and the Title Registry. The Core Messaging Platform allows those entities to securely exchange business documents through the Internet. The purpose of the Title Registry is to record and transfer the rights and obligations described in a *bill of lading*, which is a legal document that contains information about the items or rights being transferred from one party to another.

5.2.4.1 The Core Messaging Platform

The Core Messaging Platform is the hub of the bolero.net system, providing document exchange capabilities between all users and applications. The platform employs security measures such as encryption and digital signatures. In addition, it ensures document originality by taking fingerprints of documents when they are submitted for transmission. After the document traverses the platform until its destination, but before being actually delivered, the platform checks the document against the recorded fingerprint. If they do not match, the platform will reject the document. The Core Messaging Platform also guarantees delivery of documents by means of a special-purpose messaging protocol that acknowledges every exchange. When a user sends a message, the Core Messaging Platform acknowledges receipt immediately and then forwards the message to its destination. When the message is delivered to the destination, the platform acknowledges the original sender again. Otherwise, the platform will inform the sender that the message could not be delivered within a certain time period.

5.2.4.2 The Title Registry

The messaging platform maintains a log of all exchanged messages, which is kept in a special operations center. Messages are stored for several years and can be used in case of dispute between trading parties. This log is essential in supporting the Title Registry. The bolero.net framework assumes that most trading transactions will require a bill of lading, which will be created, transferred, amended, and surrendered. When the transaction is complete, rights and obligations from one trading party have been transferred to another. The Title Registry records these transactions and automatically acknowledges all affected parties. The security features of the Core Messaging System are absolutely fundamental to ensure the reliable operation of the Title Registry.

5.2.4.3 The settlement process

Besides the Core Messaging System and the Title Registry, bolero.net includes value-added services. The most important of these services is SURF (Settlement Utility for managing Risk and Finance), which was designed to address mainly payment processes. SURF supports common payment methods between buyer and seller, such as open account, advance payment, or credit. With the inclusion of SURF, the three components - the Core Messaging System, the Title Registry, and SURF - are able to support the whole transaction process between trading parties. According to bolero.net, this process is called the *settlement process*. The settlement process comprises nine main steps and it involves not only buyer and seller, but also their banks, an inspector, and the carrier's import agent, as shown in figure 5.19. All participants communicate through the Core Messaging Platform.

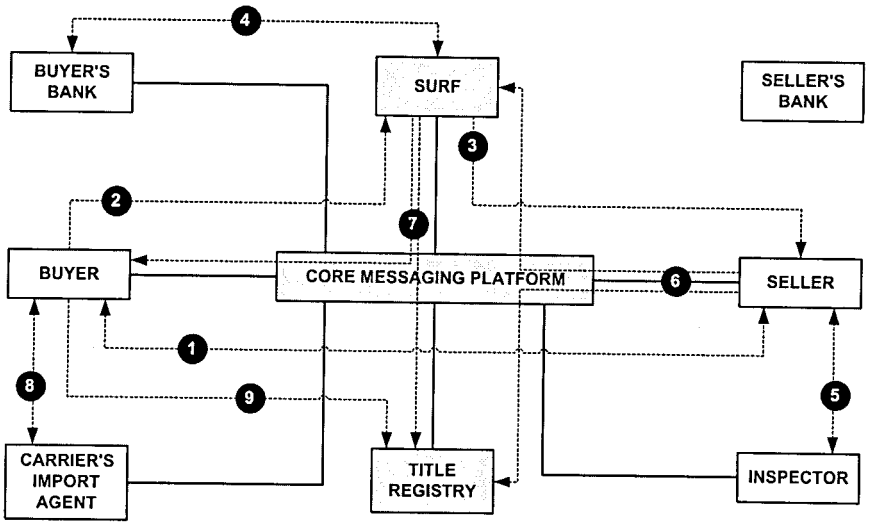


Figure 5.19: Participants in the settlement process

The settlement process starts with a commercial agreement between buyer and seller. The establishment of the agreement is complete when the buyer sends a purchase order to the seller, and the seller returns an order confirmation (step 1). From this agreement, one of the trading parties will write a formal proposal: the SURF proposal. This proposal contains information about the parties in the agreement, the goods being traded, the required trading documents, and the payment method. The buyer will write the SURF proposal and submit it to SURF (step 2), and SURF will forward the proposal to the seller for acceptance (step 3). Having received the SURF proposal, the seller can accept it, reject it, or make a counter proposal. If the seller accepts the proposal, the proposal becomes a SURF agreement. SURF will notify the buyer of the seller's acceptance.

Once the SURF agreement has been established, SURF will automatically request a *payment undertaking* from the buyer's bank (step 4). A payment undertaking is a promise to pay when certain conditions (e.g. the delivery of goods) have been met. The bank will have a fixed number of days to return the conditional payment undertaking. When that happens, SURF will check if it complies with the SURF agreement before it is sent to the seller. Once the seller receives the payment undertaking, the seller must prepare a set of documents to prove that the goods comply with the buyer's requirements. For this purpose, the seller may have to request a certificate from an external inspector (step 5). Once the seller collects all documents, including the inspector's certificate and a bill of lading, these documents are sent to SURF and to the Title Registry (step 6). The Title Registry will assign the ownership of the bill of lading to the seller.

SURF will check the documents against the agreement and, if there is any discrepancy, a report is sent either to the seller to perform changes, or to the buyer and the buyer's bank to accept the discrepancies. If there is no discrepancy, SURF will notify the seller and the buyer's bank of document compliance. It will also release the documents to the buyer and to the Title Registry (step 7). This time, the Title Registry will assign the ownership of the bill of lading to the buyer. The transfer of funds then takes place between the buyer's bank and the seller's bank. Once the goods arrive at their destination, the carrier's import agent will notify the buyer (step 8). To receive the goods, the buyer must surrender the bill of lading to the carrier's import agent. For this purpose, the buyer sends the

bill of lading to the Title Registry (step 9), which will assign its ownership to the carrier's import agent. In return, the carrier's import agent issues a delivery order to the buyer, enabling the buyer to pick up the goods.

5.2.4.4 boleroXML

The settlement process that bolero.net addresses requires several kinds of documents to be exchanged between autonomous participants. In order to facilitate interoperability between these participants, bolero.net has developed a set of standard document formats. This set of document formats is called boleroXML and it defines about 80 standard document types ranging from commercial, transport, and certification of goods to insurance, banking, and customs documents. Incidentally, the bill of lading is defined as one of the transport documents. The boleroXML document specifications were developed under the assumption that the document standards required by bolero.net did not already exist, so the boleroXML document types have been developed regardless of other standards or frameworks. More recently, bolero.net representatives have shown interest in submitting boleroXML as a contribution to ebXML [Bolero, 1999b].

5.2.5 ebXML

The purpose of the ebXML framework [UN/CEFACT and OASIS, 2001e] is to facilitate a single, global electronic marketplace where enterprises can find each other and conduct business by exchanging XML-based messages. Its aim is not only to enable document exchange, but to build a common repository that stores information about enterprises and their business processes. This repository, or registry as it is called within ebXML, allows enterprises to register themselves and to find potential business partners. The ebXML framework specifies what information should be published in the registry, as well as how to implement such a registry.

5.2.5.1 The ebXML registry

The ebXML registry stores information about enterprises, including, but not limited to, profile information, process definitions, XML-based document formats, UML models, and software components [UN/CEFACT and OASIS, 2001c]. In general, these information items are called *registry objects*. In addition, the registry provides a set of *registry services* that allow registry clients to access registry content [UN/CEFACT and OASIS, 2001d]. The registry services are specified as a set of interfaces that registry clients can invoke in order to modify or browse the contents of the registry. The ebXML registry provides two main services: the Life Cycle Management Service controls the life cycle of objects stored in the registry, and the Query Management Service controls the discovery and retrieval of those objects from the registry.

The Life Cycle Management Service defines four states for a registry object: the object can be submitted, approved, deprecated, or removed. The submitted state is an intermediary step between the submission of the object to the registry and its full availability to other registry clients, which will happen once the object becomes approved. The approval is to be granted by the client which originally submitted the object to the registry. The deprecated state means that the object is still available from the registry, but it is not a valid object anymore. Once an object is removed from the registry, it is no longer available to other clients. The LifeCycleManager is the single most important interface of the Life Cycle Management Service: it allows clients to submit, approve, deprecate and

remove objects from the registry, as well as to dynamically add and remove attributes from existing registry objects.

The Query Management Service allows registry clients to search for and query every kind of registry object. The ebXML registry supports two types of queries: one is based on filters and the other is based on SQL. In both types of query, the client submits a query request to the Query Management Service using the QueryManager interface. The registry will then return a query response back to the client, either synchronously or asynchronously. The query response will contain a collection of registry objects that match the specified criteria. Filter queries are expressed in an XML syntax and they allow clients to query for specific objects, for associations between objects, for the classification of objects, or for enterprises that have submitted objects to the registry, for example. SQL queries are intended to support clients that require advanced query capabilities. They rely on the fact that every registry object has a universally unique identifier (UUID); this identifier can be used in SQL select expressions.

5.2.5.2 Choreographies and collaboration activities

A distinctive feature of ebXML is that it allows enterprises to store business process models in the registry. According to ebXML, enterprises establish business collaborations with each other. Within a given business collaboration, each enterprise plays its own role in a set of document exchanges. The complete set of document exchanges between those enterprises defines a *choreography*. The ebXML requires choreographies to be described using the ebXML Business Process Specification Schema [UN/CEFACT and OASIS, 2001a]. The Business Process Specification Schema (BPSS) is able to describe choreographies either with UML activity diagrams or with a special-purpose XML format. In both versions, BPSS describes choreographies as sequences of collaboration activities, where each activity is performed by a different enterprise.

A collaboration activity is always delimited by a request message, which initiates the activity, and a response message, which completes the activity. In general, each message carries a single business document, which is referred to only by its name. Messages also contain other elements, namely security attributes, a document envelope and any optional attachments to the business document. Security attributes are used to indicate whether the document is authenticated, whether it is confidential, and whether it has been protected, with a message digest, from being tampered with. The document envelope identifies the requesting and responding activities, and it contains the name of the business document and of any additional attachments. In case of a response message, the document envelope also contains a boolean value indicating whether the message conveys a positive or negative response.

5.2.5.3 Collaboration Protocol Profiles

The set of possible choreographies described with BPSS are the basis for identifying the capabilities of an enterprise to engage in electronic message exchanges with other partners. According to ebXML, the message exchange capabilities of an enterprise are described by a Collaboration Protocol Profile (CPP) [UN/CEFACT and OASIS, 2001b]. The CPP is an XML document that includes details of message transport, security constraints, and bindings to a specific BPSS document. The main sections of a CPP document are: PartyInfo, Packaging, Signature, and Comment. These elements are shown in figure 5.20.

The PartyInfo section identifies the enterprise and describes most of its message exchange capabilities. For a large organization, the CPP may include several PartyInfo elements for different purposes.

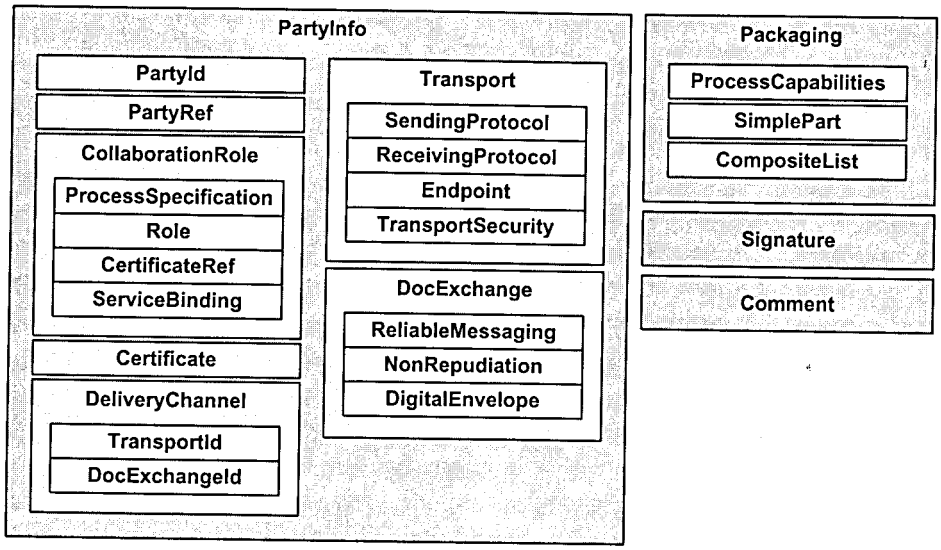


Figure 5.20: Elements of the ebXML Collaboration Protocol Profile

This element is further divided into several elements. PartyId is a logical identifier for the enterprise, to be listed in common directories such as the ebXML registry. The PartyRef element contains a link to an external site where additional information about the enterprise can be found. The CollaborationRole element associates the enterprise with a particular role in a choreography that is described by a BPSS document. For example, if the BPSS document specifies two roles such as "buyer" and "seller", the CollaborationRole element may specify this enterprise as playing the role of the buyer.

The CollaborationRole element has four sub-elements. The first sub-element, ProcessSpecification, contains the name of the BPSS document that describes the choreography. The second sub-element, Role, is one of the role names in that choreography. The third sub-element, CertificateRef, refers to a certificate listed ahead in the PartyInfo section. The fourth and last sub-element, ServiceBinding, specifies a default DeliveryChannel for all message traffic that is to be sent to this enterprise, within the context of this particular choreography. The DeliveryChannel elements are also included ahead in the PartyInfo section.

The PartyInfo section may include one or more Certificate elements. Each of these elements specifies a digital certificate as defined by the XML Signature Syntax and Processing Recommendation [W3C, 2002]. Each Certificate element is uniquely identified within the CPP so that it can be referred to from the CertificateRef sub-element in CollaborationRole. The same applies to the DeliveryChannel elements, which are referred to in ServiceBinding sub-elements.

A DeliveryChannel element is a combination of a Transport element and a DocExchange element, and it contains the requirements concerning message receipt. The PartyInfo section may contain several DeliveryChannel elements, and within each of these elements the Transport and DocExchange elements are referred to by their respective identifier. The same Transport element or the same DocExchange element may be referred to by more than one DeliveryChannel element. Additionally, and not shown in figure 5.20, each DeliveryChannel element specifies a number of security measures, such as whether a secure or insecure transport protocol should be used, whether the message should be encrypted or not, or whether it should be authenticated or not.

Within the PartyInfo section, Transport elements specify the protocols that the enterprise will use to send and receive messages. These protocols can be HTTP, SMTP, FTP and any additional protocols that the enterprise is able to use. The Endpoint sub-element specifies the communication address associated with the receiving protocol, so it may be a URL, an e-mail address or an FTP address, depending on the specific protocol being used. The optional TransportSecurity sub-element identifies the supported security protocols for message exchange, both sending and receiving. It also refers to any certificates that should be used together with those protocols.

5.2.5.4 The ebXML Message Service

The DocExchange element refers to the parameters to be used when exchanging documents through a MOM system, specifically the ebXML Message Service [UN/CEFACT and OASIS, 2002]. These parameters specify communication properties that are equivalent, in purpose, to the QoS properties of the CORBA Notification Service. The ReliableMessaging sub-element specifies the degree of reliability for message delivery: it can either require messages to be delivered once and only once, or it can specify a best effort delivery policy. The same ReliableMessaging sub-element may specify if the ebXML Message Service should check for duplicates, and if messages should be delivered in the same order they were sent. Furthermore, it specifies a maximum number of delivery attempts, a time interval between retries, and a minimum amount of time during which messages should be kept persistently. The NonRepudiation sub-element determines whether messages should be digitally signed or not, and the DigitalEnvelope element specifies an message encryption algorithm.

The ebXML Message Service specifies the message format for transmission using a communication protocol, preferably HTTP or SMTP. The specified message format, like in BizTalk, is strongly based on SOAP [W3C, 2000b]. Thus, ebXML messages have a SOAP-compliant header, and a body that carries the desired business documents. The ebXML Message Service is composed of three main parts: a message service interface, a set of message handling services, and the mapping to the underlying transport protocols, as illustrated in figure 5.21. The purpose of *header processing* is to create a message header for the original documents, taking into account unique identifiers, time stamps, and digital certificates. To do this, the Message Service relies on *security services* to create digital signatures, and to encrypt and authenticate messages. The *reliable messaging services* handle message delivery and acknowledgment, and include support for persistence, repeated delivery attempts, and error notification. The *message packaging services* wrap an ebXML message into SOAP message before it is sent using an underlying transport protocol.

Returning to the CPP structure shown in figure 5.20, the Packaging section specifies how the message header and body are to be packaged before being transmitted. In particular, the Packaging section specifies the rules to encode constituent parts (of the CPP or any other message) using MIME. The ProcessingCapabilities element specifies whether these rules apply to both message encoding or decoding, or if they should apply only to one of these operations. To enumerate the specific set of MIME parts in a message, one or more SimplePart elements must be inserted in the Packaging section. Each of these SimplePart elements is given by an identifier and a MIME content-type value. Subsequently, the CompositeList sub-element may optionally specify how to combine simple parts into MIME multiparts, or how to encapsulate them within secure MIME content-types.

Depending on the implementation scenario, the CPP may have to be signed in order to ensure its integrity. For this purpose, the Signature section contains a set of elements that may be used to insert a digital signature in the CPP. The contents of this section and of its elements are defined once again by the XML Signature Syntax and Processing Recommendation [W3C, 2002]. Finally, the CPP

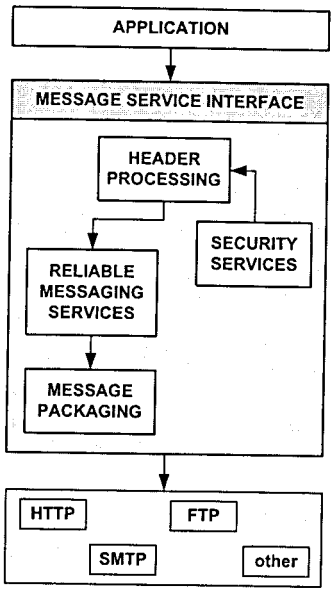


Figure 5.21: Main components of the ebXML Message Service[77]

may contain one or more Comment elements, which are textual notes that may be added to serve any purpose.

5.2.5.5 Collaboration Protocol Agreements

The CPP of each enterprise should be published in the ebXML registry, allowing enterprises to find potential business partners. Since CPPs refer to pre-defined choreographies and messaging requirements, two enterprises that intend to engage in electronic exchanges must have compatible CPPs. The two enterprises must be willing to behave according to a specific role of the choreography, and they must agree on the use of message packaging rules, security measures and transport protocols. The ebXML framework introduces a special-purpose document - the Collaboration Protocol Agreement (CPA) - that allows two companies to establish an agreement concerning these issues. The CPA has a similar structure to the CPP, except for the fact that all elements that pertain a specific enterprise must be duplicated in order to account for the other enterprise as well. Therefore, a CPA has two PartyInfo sections and two Signature sections, as shown in figure 5.22.

The first section of a CPA defines its lifetime to be within a start date and an end date. The CPA is not valid outside this time interval, but if a collaboration activity is still running at the time the end date is reached then it must be allowed to complete. The second section, the ConversationConstraints, places two limits on the interaction between parties. On one hand, it places a limit on the number of exchanges between two enterprises, so that when this limit is reached the CPA is terminated and must be negotiated again. On the other hand, it places a limit on the number of collaboration activities running simultaneously between enterprises. The consequences of reaching this limit are dependent upon a particular implementation. For example, new collaboration activities beyond this limit may be rejected, or they may be enqueued for subsequent processing.

[77][UN/CEFACT and OASIS, 2002]

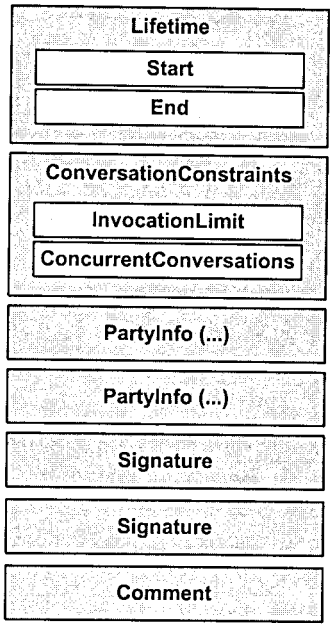


Figure 5.22: Elements of the ebXML Collaboration Protocol Agreement

The CPA has one PartyInfo section for each enterprise, which has exactly the same structure as that of a CPP. However, the ProcessSpecification sub-element in both PartyInfo sections must refer to the same BPSS choreography. In addition, the Transport elements must identify the agreed set of transport protocols that will be used in complementary roles: the sending protocol for one role must always match the receiving protocol of the party playing the other role. As for the Signature sections, the CPA specification [UN/CEFACT and OASIS, 2001b] recommends that the first party should sign the CPA contents, and that the second party should sign the CPA together with the first signature. In addition, ebXML suggests that an electronic notary may sign over both signatures.

5.2.5.6 The ebXML functional view

Figure 5.23 summarizes the use of CPPs and CPAs. First, it is assumed that each enterprise has implemented a Business Service Interface in order to enable interoperability according to the ebXML framework. This is called the *implementation phase*. Also within this phase, each enterprise will publish its CPP on the registry, binding it to an existing choreography. Alternatively, the enterprise may introduce a new choreography. As a result, the registry will contain a set business process and information models. In a second phase, enterprises begin to discover and retrieve these models in order to find potential business partners. The typical discovery method is to request CPPs of other enterprises, and to subsequently match CPPs in order to achieve CPAs. The third phase is the *run-time phase*, when enterprises exchange messages with each other using the ebXML Message Service and according to the CPA they have established. These three phases - implementation, discovery, and run-time - are the three ebXML functional phases. The ebXML registry, BPSS, CPPs, CPAs, and the Message Service are the framework components that support these functional phases.

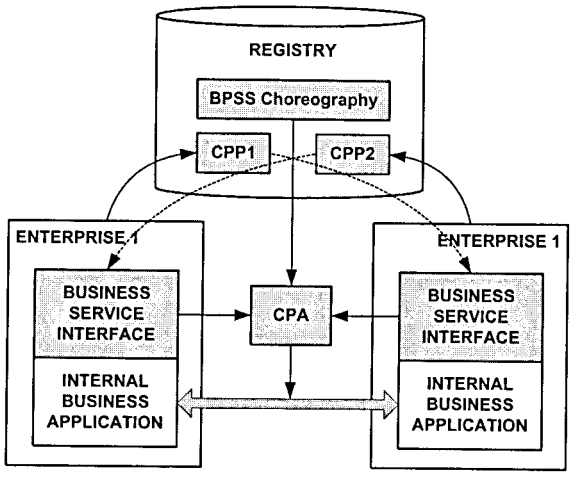


Figure 5.23: Simplified ebXML functional view

5.2.6 RosettaNet

The RosettaNet framework [Schoonmaker, 2001] is one of the most comprehensive and successful B2B frameworks. The RosettaNet framework, like the ebXML framework, employs a top-down approach requiring business partners to specify their business processes in detail before defining the XML messages they will exchange. RosettaNet includes a set of standards that aim at supporting information exchange throughout the whole supply chain, particularly within the Information Technology, Electronic Components and Semiconductor Manufacturing industry sectors. For this purpose, the RosettaNet framework defines a set of common dictionaries that establish the properties to be used in business processes between supply chain partners. These inter-enterprise business processes are called Partner Interface Processes (PIPs) and they define the message exchange between partners. The RosettaNet Implementation Framework specifies the packaging, routing and transport details for these messages.

5.2.6.1 RosettaNet dictionaries

In order to ensure that partners can understand each other when they refer to product and business items, the RosettaNet framework specifies two kinds of dictionaries: one is the *technical dictionary* and the other is the *business dictionary*. Both dictionaries define a common name to be used when referring to particular items. The technical dictionary defines product-related information items ranging from physical characteristics, such as length and width, to manufacturing characteristics such as operating temperature and input voltage. Clearly, these properties were defined for the specific industry sectors that RosettaNet applies to. For different industry or service sectors, new technical dictionaries would have to be written. Regarding the business dictionary, the properties defined in this dictionary are more widely applicable: they concern business-related information including, but not limited to, marketing, sales, customer, billing, and shipping information. The purpose of both the technical and business dictionaries is not to clarify information items, but to establish the set of standard, commonly understandable properties that business partners must use, instead of their own denominations, when performing exchanges.

5.2.6.2 Partner Interface Processes (PIPs)

According to the RosettaNet framework, B2B exchanges take place within the scope of Partner Interface Processes (PIPs). The purpose of PIPs is to define how two business processes, running at different enterprises, are linked and synchronized by means of message exchange. RosettaNet refers to this linkage as *process alignment*, which is the aim of the whole framework. Each PIP defines an interaction procedure between two or more business partners, i.e., it defines both the internal activities that take place at each business partner and the message exchange between partners. However, some interactions are related to each other even though they are not directly connected. For example, the request for a quote may lead to a new order, but this does not mean that both actions are directly connected, e.g., there may be an internal order approval process between the two. RosettaNet defines one PIP for each interaction and then it groups related PIPs into segments.

However, RosettaNet realizes that there are many business processes, pertaining to different business areas, that must be aligned between business partners so, besides grouping PIPs into segments, it groups segments into eight different clusters [RosettaNet, 2000]:

- Cluster 0 is devoted to administrative functionality, and it has two segments that contain PIPs for error notification and testing purposes, respectively.
- Cluster 1 contains two segments that concern partner information and product information, respectively. The first segment includes those PIPs that allow partners to maintain contact, billing and shipping information about each other. The second segment contains PIPs that allow buyers to subscribe product information from suppliers.
- Cluster 2 concerns technical information and it contains those PIPs that allow a supplier to notify changes in product characteristics, and to exchange product design and engineering information with other business partners.
- Cluster 3 is devoted to order management and it includes all PIPs that deal with interactions such as requests for quotation, product configuration, new orders, order shipping, invoicing, and payment.
- Cluster 4 concerns inventory management and it contains PIPs that basically handle notifications which are relevant to collaborative forecasting, inventory and sales reporting, inventory replenishment and price changes.
- Cluster 5 is called marketing information management but it contains mainly those PIPs that allow product distributors to receive incentives and to request shipping and payment authorizations from suppliers. These PIPs apply only to the Electronic Components industry sector.
- Cluster 6 handles service and support, i.e., it concerns those PIPs that allow buyers to register their products, to establish warranty contracts, and to request technical support from suppliers.
- Cluster 7, which is a recent addition to RosettaNet, is called manufacturing and it contains a set of PIPs that enable the exchange of design, process, quality, and manufacturing floor information in order to establish what RosettaNet calls a "virtual manufacturing environment".

Every PIP is defined as an inter-enterprise business process, which typically comprises the exchange of request messages and response messages between two business partners. Some PIPs may involve



more than two business partners, they may require exchanges other than request-response, or only a request message without response, but the typical PIP definition is that shown in figure 5.24. A PIP can be initiated according to one of two possibilities: either it begins from the need for certain information such as the need for product marketing information, or it begins from the generation of information such as sales promotions or an engineering change. This information need or generation is first processed internally before being submitted as a request message to the other business partner.

As soon as the request is received at the other end, it enters another series of internal activities. Eventually, the internal processing of the request at the second business partner will produce a response message to be sent back to the first partner. This response message contains either the desired information (if the PIP was initiated due to an information need) or a confirmation of the actions undertaken (if the PIP was initiated due to a engineering change, for example). The PIP defines the XML message structure for request and response messages, as well as the business properties (taken from the technical and business dictionaries) to be used within each message.

5.2.6.3 The RosettaNet Implementation Framework

The request message, the response message, and any other message that two partners may exchange must follow a common format known as the RosettaNet Business Message. Furthermore, these messages must be packaged and transmitted using a transport protocol, and they must employ some security measures. The RosettaNet Implementation Framework [RosettaNet, 2002a] is a technical specification that addresses these details. The RosettaNet Implementation Framework begins with the specification of the RosettaNet Business Message, as shown in figure 5.25. RosettaNet Business Messages are composed of several XML documents, except for the attachment fields which may contain any application-specific data. The RosettaNet Implementation Framework specifies that these elements must be encoded separately in a multipart MIME message.

The preamble header is an XML document on its own, and it used to identify the standard and version of the standard that the message is compliant with. It also includes the code of an administration authority that plays the role of administrator in Cluster 0 PIPs. The delivery header is another

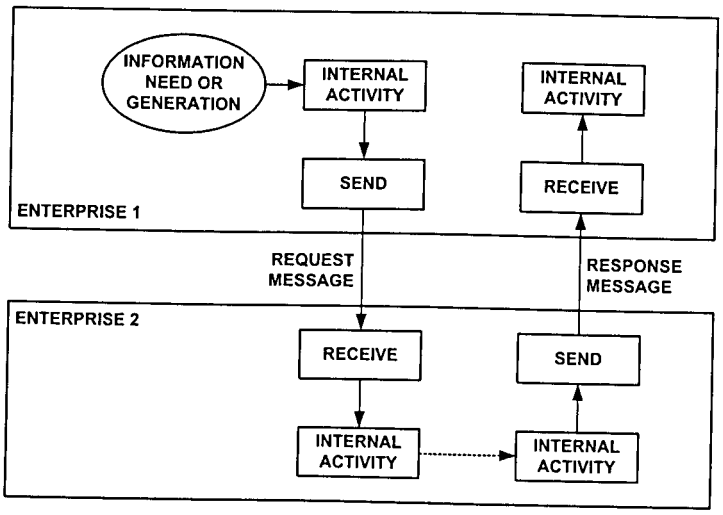


Figure 5.24: Typical PIP definition

XML document that contains several parameters concerning message routing: a time stamp, a message identifier, whether or not the message requires secure transport, the identification of sending and receiving ends, and an indicator that can be used to convey an affirmative or negative answer. The service header relates the message with a particular PIP. For this purpose, the service header includes a PIP code and it contains the business roles of the sending and the receiving partners; these roles are defined in the PIP being used. Moreover, the service header identifies the activity running at the sending partner, it indicates whether or not the message is a reply to a previous message, and it describes the attachments included in the message payload.

The message payload contains the PIP message being sent from one partner to another, such as a request for quote or a purchase order. This message is encapsulated within the "service content" field, which is a separate XML document whose format is defined by the particular PIP being used. The RosettaNet Implementation Framework requires PIP messages to be acknowledged. The purpose of this acknowledgment is to notify the original sender that the PIP message is being handled at the receiving end. The acknowledgment messages are a second type of messages, called *signal messages*, which the RosettaNet Implementation Framework distinguishes from PIP messages. Signal messages may carry positive or negative acknowledgments. A positive acknowledgment means that the message was received and that it was considered to be valid. A negative acknowledgement means that either the message was found to be invalid, or that there was a run-time error when performing the requested action. The "service content" field in figure 5.25 may contain a PIP message or a signal message expressing a positive or negative acknowledgment.

The RosettaNet Implementation Framework requires the use of S/MIME to protect RosettaNet Business Messages. It also requires messages to be signed using digital certificates in order to support message authentication and non-repudiation. After the Business Message is packaged into a S/MIME message, the RosettaNet Implementation Framework allows the message to be transmitted using HTTP or SMTP. In the first case, and if the message requires transport security, HTTP may be used together with SSL. In the second case, transport security cannot be employed. RosettaNet points out that other transport protocols may be adopted without changing the message format.

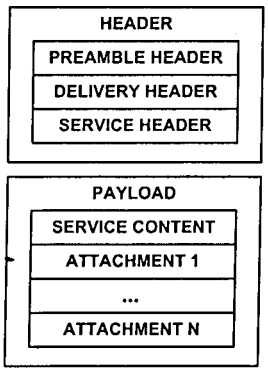


Figure 5.25: The RosettaNet Business Message

5.2.6.4 Trading Partner Agreements

Besides the dictionaries, the PIPs and the RosettaNet Implementation Framework, RosettaNet includes a set of projects (called programs) to facilitate partner collaboration. One of such programs is the development of a standard for Trading Partner Agreements (TPAs). The TPA program [RosettaNet, 2002b] has created a multipurpose structure for TPAs, which comprises five modules. The first module describes general legal provisions including, but not limited to, terms and conditions, security measures, and the liabilities of each partner. The second module is a non-disclosure agreement, which establishes the confidentiality of the exchanged information. The third module is entitled "portal services" and it contains contact information about each party, together with the Internet address and access rules for Web portals that one or both parties may provide. The fourth module is called "XML services" and it defines the B2B framework that both parties agree to use. In case such framework is RosettaNet, this module refers to particular PIPs. As an alternative to the fourth module, business partners may instead sign a fifth module, "EDI services", to specify the EDI standards they will use when performing exchanges.

5.2.6.5 RosettaNet in action

The RosettaNet framework is perhaps the B2B framework enjoying more success. Several members of the RosettaNet consortium have been performing exchanges based on this framework [Schoonmaker, 2001] and the RosettaNet consortium claims that in early 2002 there were more than 450 partner connections worldwide using RosettaNet [Kak and Schoonmaker, 2002]. Incidentally, the BizTalk-compliant server supplied by Microsoft now supports RosettaNet, including all published PIPs and the RosettaNet Implementation Framework.

The success of RosettaNet is probably due to two main reasons. The first is that RosettaNet has a narrow industry focus, which gives it a more manageable scope than other frameworks that often try to be as generally applicable as possible. By focusing on very specific industry sectors, RosettaNet was able to identify and document typical business processes for those industries. This is in clear contrast with other frameworks, which either assume pre-defined business processes (such as OBI or bolero.net) or leave all process modeling up to the implementation (such as BizTalk and ebXML). Because RosettaNet has such a narrow industry focus, it was able to define typical PIPs while leaving room for defining additional ones. As a result, RosettaNet may be less general than other frameworks, but it can be promptly implemented in the industry sectors it applies to.

The second main reason that explains the success of RosettaNet is that it attempts two types of integration simultaneously, whereas other frameworks only focus on one of them. On one hand, RosettaNet introduces dictionaries that establish common information items. This allows trading partners to exchange mutually understandable business documents, which is also the goal of other B2B frameworks that focus on standardizing document formats. But then, on the other hand, RosettaNet also introduces PIPs to align business processes between business partners. This is the same goal of yet other frameworks that focus on enabling message exchange rather than on particular document formats. The unique characteristic of RosettaNet is that it combines both these goals and specifies standardized approaches regarding both the structure of information and the interaction processes between business partners.

5.2.7 tpaML

Several B2B frameworks have realized the need to express agreements between business partners. For example, cXML introduces the concept of master agreements, ebXML specifies how to build Collaboration Protocol Agreements (CPAs), and RosettaNet has a created a project to standardize Trading Partner Agreements (TPAs). The tpaML language [Sachs and Ibbotson, 2000] is the basis of a B2B framework which focuses exclusively on *executable trading partner agreements* [Dan et al., 2001]. According this framework, a trading partner agreement (TPA) is a document that describes the terms and conditions for the interaction between two business partners. A set of several related interactions between any two parties is referred to as a *conversation*. The main issue about executable TPAs is that they allow business partners at both ends to automatically generate code that will implement the interactions between them. Executable TPAs are written with tpaML, an XML-based language that can be processed by a special-purpose code-generating tool. Because an executable TPA describes the interactions that will take place between two parties, each party maintains complete independence from the other party with respect to implementation details and internal business processes.

5.2.7.1 Structure of executable TPAs

An executable TPA is an XML document which can be divided into several main sections, as shown in figure 5.26. The first of these is the TPAInfo section, which is a preamble that contains general information about the TPA and about the parties involved. An executable TPA may describe a set of interactions according to a B2B framework that specifies the document formats and the message exchange sequence between business partners. For example, it is possible to define an executable TPA for business partners that interact according to OBI [Sachs et al., 2000][78]. In this case, the TPAInfo section includes the name and version of the protocol being used, and possibly the roles played by each partner. The main purpose of the TPAInfo section, however, is to provide information about the business partners and, if applicable, also about an arbitrator that may be designated in order to solve disputes. The TPAInfo section may contain additional information such as the lifetime of the TPA and the maximum number conversations before the TPA has to be re-negotiated.

The Transport section contains information about the network transport protocols that the aforementioned parties can use to communicate with each other. Usually, this section will specify the details that allow parties to communicate via an Internet protocol such as HTTP, FTP or SMTP; alternatively, it is possible to specify EDI or even a proprietary message queuing system as the transport. The parties may require messages to be encrypted and authenticated; message encryption is assumed to be based on public key certificates, while message authentication may be implemented by means of password authentication or certificate authentication. If the chosen transport protocol can provide these security capabilities then these details may be set in the Transport section. Otherwise, the security details must be set in the DocExchange section.

The DocExchange section specifies the security details for each message, regardless of transport protocol. In principle, the DocExchange is only used to specify whether a digital signature should be appended to each message and, if so, which kind of certificate and signature algorithm should be used. In addition, if parties require features such as encryption and the underlying transport protocol does not support them, then they may specify the required security features in the DocExchange section. It

[78] There are slight differences between [Sachs and Ibbotson, 2000] and [Sachs et al., 2000] with respect to the structure of tpaML documents. The description in this section follows the tpaML specification [Sachs and Ibbotson, 2000].

should be noted that all security operations specified in the DocExchange section will be performed before the message is passed to the transport protocol for transmission.

The tpaML language introduced the concept of *delivery channel* to refer to a combination of a Transport definition and a DocExchange definition which, together, specify how messages are transferred from one party to the other. It is interesting to note that this concept of delivery channel is precisely the same as in ebXML, as shown in figure 5.20 on page 196.

5.2.7.2 Service interfaces

The BusinessProtocol is the most important section in the executable TPA because it describes the interactions that the TPA applies to. An executable TPA can describe one or more possible interactions between one party that plays the role of *client* and another party that plays the role of *server*. A given party may play both server and client roles in different interactions. Each possible interaction is called

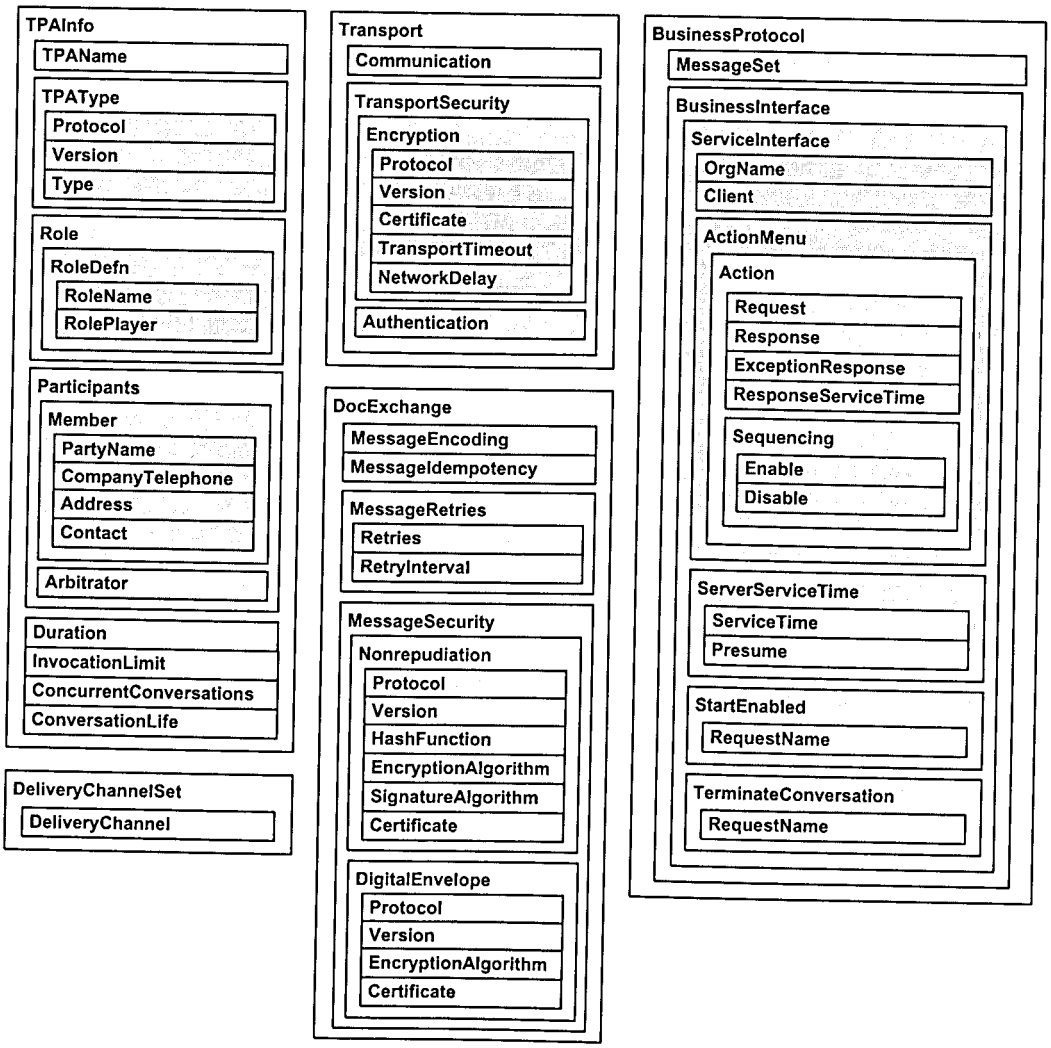


Figure 5.26: Structure of tpaML documents

a *service interface*, and it is always described from the perspective of the server. Figure 5.26 illustrates the structure of ServiceInterface elements within the BusinessProtocol section.

A service interface contains a list of *actions*, which represent units of work that the server is able to perform in reaction to a certain *request*. Each action may or may not return some result information; this information is sent back to the client as a *response*. Before the server issues the response containing the final result, it may produce several intermediary responses; in general, an action may produce any number of responses (including zero). An action may also produce an exception response in case of error or other unusual condition.

When the server receives an action request, it may have a maximum amount of time to produce a response. This time limit can be specified in one of two ways: either it is defined for each action or it is defined for the whole service interface. In case both definitions are present, the action-specific parameters override the service interface parameters.

In general, if a service interface has several actions then they are performed sequentially according to the same order in which they have been defined. Alternatively, it is possible to specify sequencing rules that determine which actions are allowed and which actions are not allowed after a given action has been performed. The actions that are allowed after a given action completes are referred to as being *enabled*, whereas the actions that are not allowed are considered to be *disabled*. Both enabled and disabled actions are identified by the name of the request that initiates them.

Each action may have its own list of enabled and disabled actions; this list is only effective after the action has been successfully completed. The fact that an action is enabled does not necessarily mean that it will be executed. An action is only executed when the client issues the request for that action. Disabled actions cannot be executed even if the client issues the corresponding request.

For each service interface, there must be one or more actions that can be requested initially before any other action has been executed; these actions are enabled when the conversation starts (the StartEnabled element). Conversely, there may be some actions after which no further actions can be requested; these actions terminate the conversation. The actions that are initially enabled and those that terminate the conversation can be specified within the service interface, and they complete the sequencing rules for an executable TPA.

5.2.7.3 Code generation from executable TPAs

There are two Java-based software tools that support the use of executable TPAs [Sachs et al., 2000]. One is an authoring tool, which allows the user to easily prepare a tpaML document. This document may be created from scratch, or it may be assembled from elements that have been previously prepared. For example, by the time a user creates a TPA, the definitions for the network transport protocols could be already available; the user could then insert them directly in the TPA's Transport section.

The second tool is a code-generation tool, which can create executable code from a given tpaML document. Both parties should use this tool in order to create the executable code that implements both ends of a conversation. For each service interface, the code-generation tool creates one TPA object for the client and one TPA object for the server: the client-side TPA object translates local method calls into request messages that are sent through the network transport protocol, and the server-side TPA object translates request messages received via the network transport protocol into local method calls. The purpose of the code-generation tool is somewhat equivalent to that of a CORBA IDL compiler, which creates skeletons (equivalent to server-side TPA objects) and stubs (equivalent to client-side TPA objects) from an IDL file (equivalent to a tpaML document). The main

difference is that, because a tpaML document specifies a set of sequencing rules, a TPA object knows which actions are allowed and which are not allowed after a given action has been invoked.

5.2.8 eCo

The eCo architecture [CommerceNet, 1999] is not a concrete framework, but rather an abstract architecture for B2B frameworks. The main purpose of the eCo architecture is to enable interoperability between heterogeneous e-business systems, which may be based on different B2B frameworks. The eCo architecture realizes that e-business system implementations vary widely and, instead of restricting them to a specific framework, the goal of eCo is to allow each enterprise to define and expose meta-data descriptions of its e-business system. These meta-data descriptions allow interested parties to understand the e-business system and to configure their resources in order to interoperate with this system.

In essence, the eCo architecture specifies how to build an XML file, to be placed at the root of a Web site, that describes the interoperability capabilities of the e-business system available on that Web site. Several queries can be performed on this Web site in order to retrieve information about the interoperability capabilities of the e-business system. These HTTP queries are performed by submitting requests to different URIs. All these requests, regardless of the URI they refer to, lead to responses that are based on information contained in the eCo-compliant XML file.

5.2.8.1 The eCo architecture layers

According to the eCo architecture, there are several levels of interoperability. These levels correspond to the seven layers of the eCo architecture, as depicted in figure 5.27. Basically, the top layer - *networks* - refers to the virtual communities that develop within a network such as the Internet, where several *markets* may exist. Within *markets*, there are several *businesses*, each one providing certain *services*. Each *service* requires a set of *interactions* through which *documents*, containing *information items*, are exchanged between business partners.

The purpose of the network layer is to locate markets and trading communities in a network. The market layer is a common portal or entry point, where businesses offer its products or services in a shared environment with other related businesses. The business layer concerns the information about each member in the trading community, be it an individual, an enterprise or a department of a larger organization. The business layer also lists the interoperability services that these members implement or require. The service layer provides meta-data about these services, i.e., it describes the requirements that will enable two business partners to interact. A service can be regarded as an interface to an internal business process. The interaction layer addresses the set of interactions that can be performed with a given service. For example, it may specify the set of request-response interactions that the service supports. The document layer refers to the specific business documents that are exchanged within each interaction with a service. In the case of a request-response interaction, the document layer defines both the request and response documents. These documents, which in general will be XML documents, are composed of several markup elements that are used to encapsulate individual information items.

5.2.8.2 Published interfaces

Each layer in the eCo architecture has its own *published interface*. A published interface is a mechanism that enables other parties to retrieve, using HTTP, information about the interoperability capabil-

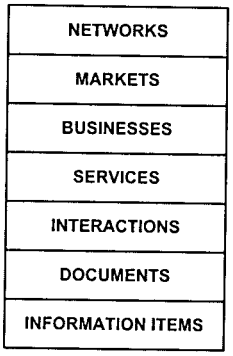


Figure 5.27: The eCo Architecture layers

ities within a certain architecture layer. A published interface contains a set of queries that correspond to HTTP queries, i.e., each interface query is represented by a different URI. In response to a query on such URI, an eCo-compliant Web site must return an XML document containing the answer to that query.

For example, the published interface for the network layer contains one query called NetworkGetMarkets. This query returns a list of markets that exist within the network. On the other hand, the published interface for the market layer contains one query called MarketGetBusinesses, which returns a list of businesses that participate on this market. In general, the published interface for each layer exposes a set of queries that allow a remote party to retrieve information about the next lower level. As a result, queries can be subsequently performed until all information about the available services, interactions documents and information items has been retrieved.

Figure 5.28 shows the published interface for each architecture layer. Each published interface is composed of a set of queries whose name begins by a prefix that identifies the layer they belong to. Despite the difference in their names, some queries perform essentially the same function. For example, the first query of every published interface returns a set of properties concerning the corresponding layer. The second query of every published interface allows a remote party to retrieve a list of references to elements of the following (lower) level.

In all published interfaces, except for the network layer, the last method allows to proceed backwards from a lower level into an upper level. However, most of these queries are optional, i.e., the eCo architecture does not require them to be implemented in order to consider that an e-business system is compliant with the eCo architecture. In fact, only the implementation of the first query from each published interface, shown in gray in figure 5.28, is mandatory. In addition, the eCo architecture does not require the implementation of all published interfaces. For an e-business system to be minimally compliant with eCo, only the published interface of the business layer must be implemented.

Figure 5.29 illustrates the published interface for the business layer together with the response document for the BusinessGetProperties query. The BusinessGetProperties query returns an XML document, which is called a BusinessPropertySheet. The BusinessPropertySheet contains the set of properties shown in the right side of figure 5.29. The name property contains the legal name of the enterprise, and the general address properties contain the contact information for this enterprise. The technical address properties are used to place technical questions about the operation of the e-business system. The TermsAndConditions property is a pointer to another URI containing legal information

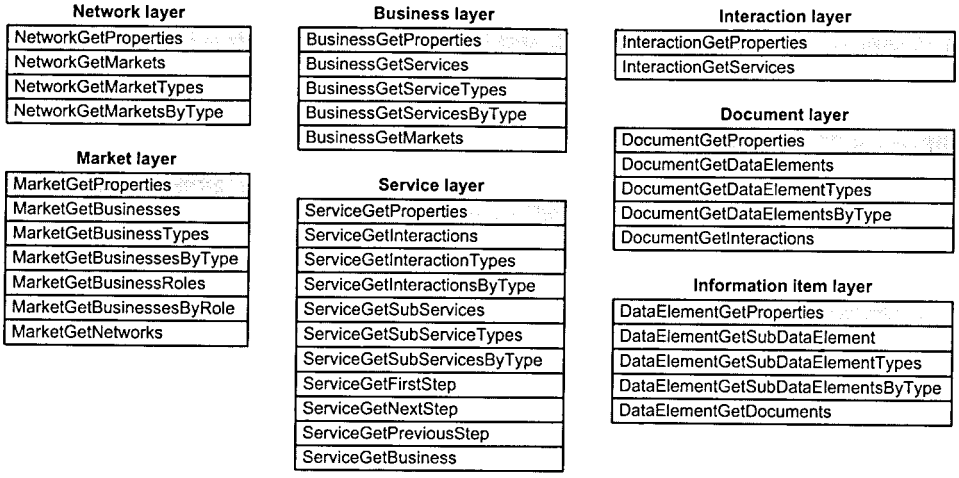


Figure 5.28: The published interface for each eCo architecture layer

about how this enterprise is willing to trade with other enterprises. The Credentials property contains a digital certificate of the enterprise.

The ProcessDefinitionLanguage property contains the name of a process modeling language that was used to create a process definition, which is referred to by the following property. The eCo architecture does not specify or require any process modeling language in particular, leaving this issue up to the implementation, or up to the market where the business operates. The ProcessDefinition property is a reference to one or more external URIs that contain the choreographies involving the enterprise and other parties. These choreographies are high-level descriptions of general behavior between enterprises, rather than the actual document exchanges between local and remote services. The detailed interactions between these services can be retrieved from the published interface of the

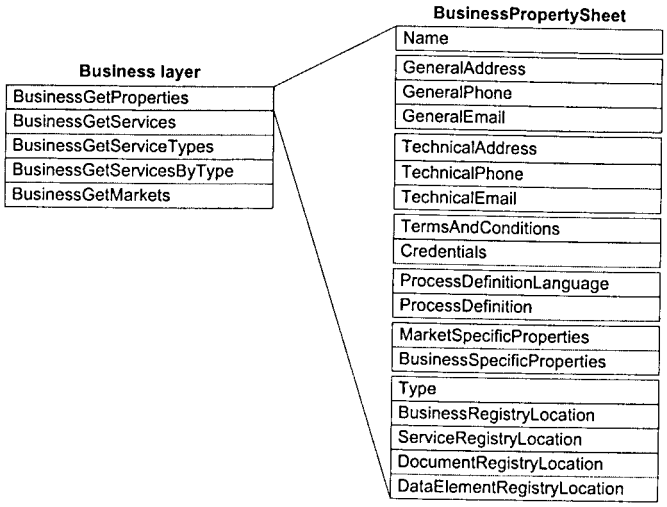


Figure 5.29: Response to the BusinessGetProperties query

service layer. Once again, however, eCo does not require the use of any particular process modeling language at that layer.

The following properties in the BusinessPropertySheet shown in figure 5.29 are market- and business-specific properties. Each of these elements is a list of URI pairs. The first URI of each pair identifies the market or business, respectively, whereas the second URI refers to a related set of properties. Market-specific properties are defined by the market where the business operates, and they describe the enterprise in a way that is meaningful only within a particular market. Business-specific properties are defined by the enterprise itself, which allows the enterprise to extend the set of properties it provides to other parties.

5.2.8.3 eCo type registries

The last set of properties in the BusinessPropertySheet shown in figure 5.29 are due to the use of *type registries*. Besides associating a published interface with each layer, the eCo architecture also associates a different type registry. Except for the network layer, all other layers have a corresponding registry: there is the market registry, the business registry, the service registry, the interaction registry, the document registry, and the information item registry. Each type registry contains information about sets of types that are appropriate for a given architecture layer. For example, the business registry contains information about several types of businesses; the service registry lists the types of services available from a particular business; the interaction registry describes, for a given service, the available interaction types (the protocols that conduct document exchanges) and the available message types (the formats that documents must follow).

The use of type registries for each layer explains most of the queries in the published interfaces shown earlier in figure 5.28. All queries that refer to layer-specific types effectively retrieve information from the corresponding type registry. In general, the response to the first query of each published interface returns a set of properties that include the location of one or more type registries. This explains the presence of the last four properties in the BusinessPropertySheet shown in figure 5.29. Each of these properties contains the URI of a different type registry. These URIs allow a remote party to directly query those registries for layer-specific type information. The type property shown in figure 5.29 is also related to type registries: for a given enterprise, it refers to the type of its business and to any other type registries where this business is listed.

The eCo type registries have two important features. The first is that types can be organized hierarchically as a tree, in which type definitions can be refined through the addition of sub-types. This hierarchy is restricted to single inheritance. The second feature is that the eCo type registries, like their corresponding architecture layers, also have their own published interfaces. But whereas these published interfaces are different for each network layer, the published interfaces for type registries support a common set of queries. These queries are shown in figure 5.30.

In a similar way to the published interface for each architecture layer, so too the first query of the published interface for type registries returns a set of properties. These properties include, but are not limited to, the registry name and URI, the name of the implementation-specific type definition language, a digital certificate for the registry, and an optional set of user-defined properties. The other queries allow a remote party to navigate through the hierarchical relationships of type definitions within a type registry by retrieving, for each type definition, its parent, child or sibling definitions. The RegistryIsDescendant query determines whether or not two type definitions are hierarchically related. The RegistryGetNodeDefinition query returns the properties of a particular type definition. These properties are: an identifier, the URI for the published interface of the business that registered

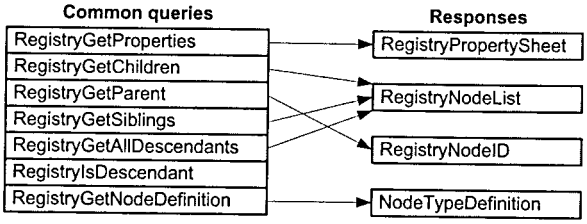


Figure 5.30: Common queries for eCo type registries

the type definition, a natural language description, the URI for an XML document format, and an optional set of user-defined properties.

In addition to the common queries shown in figure 5.30, each type registry introduces layer-specific queries. The one and single purpose of these layer-specific queries is to retrieve the root type definition of the corresponding registry. For example, the published interface for the business registry contains one additional query called BusinessRegistryGetBusinessTypeRoot, which returns the root node for the tree of business type definitions. As a second example, the published interface for the interaction registry contains two additional queries: one for retrieving the root node for the tree of interaction types and another for retrieving the root node for the tree of message types.

5.2.8.4 Agent-based e-commerce

A distinctive characteristic of eCo when compared with other B2B frameworks is that the eCo architecture specifies the way e-business systems can describe their own interoperability capabilities. This means that a remote party may retrieve this information and analyze it in order to configure its own applications to interact with this system. Alternatively, a sufficiently intelligent application may be able to retrieve that information and use it to configure itself in order to interact with the e-business system. Such an application can be called an *intelligent software agent*. Presumably, this software agent would be able to interoperate with any e-business system, provided that the interoperability capabilities of those systems are appropriately described by means of a framework such as eCo. Thus, it can be said that the eCo architecture supports agent-based e-commerce [Glushko et al., 1999].

5.3 Intelligent software agents

Intelligent software agents represent a new and rapidly developing area of research. Particularly in e-commerce [Brenner et al., 1998], telecommunications [Weihmayer and Velthuijsen, 1998] and manufacturing [Shen and Norrie, 1999a], agent-based systems are being proposed as the next generation approach for software applications. Still, there has been much discussion about what an agent is and how agents differ from regular software programs. Software agents have been regarded as online assistants for e-commerce [Brenner et al., 1998], as intelligent building blocks for information systems [Huhns and Singh, 1994], and even as a new software engineering methodology [Jennings and Wooldridge, 1999].

Despite the wide range of conceptions, there is a common understanding that agents exhibit a unique set of characteristics. These characteristics introduce a sharp distinction between agent-based

systems and other software development approaches. Most authors claim that an intelligent software agent exhibits the following properties [Wooldridge and Jennings, 1995]:

- *autonomy* - the ability to operate and interoperate with other systems without requiring intervention of humans, and to control its own actions and internal state;
- *reactivity* - the ability to perceive its environment, whether the physical world or a network environment such as the Internet, and to respond to changes on that environment;
- *pro-activeness* - the ability to act according to its own initiative, and to exhibit a goal-directed behavior;
- *social ability* - the ability to interact with other agents by means of an agent communication language.

Other authors point out slightly different properties, but they represent essentially the same characteristics. For example, some authors refer to *situatedness* [Sycara, 1998a] as the ability of an agent to receive input from its environment and to react to changes in that environment. Hence, this property has essentially the same meaning as reactivity. Other authors refer to the property of being *adaptive* [Hendler, 1999], which implies exhibiting not only reactivity but also some learning behavior.

The ability of agents to accumulate experiences and learn from their environment is an opportunity for employing artificial intelligent techniques. For example, some authors [Shepherdson et al., 1999b] describe agents that employ *support vector learning* [Schölkopf et al., 1998]. Two other properties of agents are also likely to benefit from artificial intelligence. One is their autonomy, which can only be implemented with some level of intelligence, since autonomy implies the ability to operate without human intervention. The other is social ability, which assumes that agents are able to communicate and collaborate with each other in order to achieve a certain goal. The social ability of agents, or the ability to *collaborate* [Nwana and Ndumu, 1996] as other authors call it, is one of the consequences of the fact that the concept of software agent has arisen from a field of artificial intelligence called *distributed artificial intelligence* (DAI) [Sycara, 1998a].

5.3.1 Types of agents

Software agents can be classified according to different perspectives. One of the main distinctions between agents is to classify agents as being *deliberative*, *reactive* or *hybrid* [Wooldridge and Jennings, 1995]. Usually, agents are regarded as a knowledge-based system which can sense and act upon its environment. In order to achieve this kind of reactivity, it is often assumed that agents must possess and operate over a symbolic representation of their environment. This kind of agents are called deliberative agents: they are supposed to be able to translate the real world, or part of it, into a symbolic description, and they are also supposed to reason about that information. Therefore, deliberative agents are the most akin to artificial intelligence.

Requiring agents to be able to build and use a symbolic representation of their environment is a considerable challenge, especially given that symbolic representations of reality is one of the areas that artificial intelligence has been studying since its inception. Other authors, however, argue that it is possible to exhibit intelligence without having to devise such representations [Brooks, 1991]. Intelligent behavior should be possible without symbolic representations or abstract reasoning of the kind that artificial intelligence proposes. This postulate has led to the concept of reactive agents,

which make decisions based on sensory inputs rather than on symbolic representations. Reactive agents, thus, make decisions and perform actions based on a limited amount of run-time information.

In practice, software agents often display both deliberative and reactive features. Reactive features are employed whenever there is the need to respond quickly to environmental events. Deliberative features come into play when goal-oriented plans of action must be devised, requiring the use of symbolic representations of the environment. As a result, deliberative features deal with information at a higher level of abstraction than reactive features. This suggests that it should be possible to develop layered agent architectures, where each layer corresponds to a higher level of abstraction. This has been the approach, for example, in Touring Machines [Ferguson, 1992] and InteRRaP [Müller and Pischel, 1993], both of which are layered agent architectures.

5.3.1.1 Mobile agents

A second perspective for classifying agents is to determine whether they are *static* or *mobile* agents. By definition, mobile agents are those agents that are able to move around within a network, such as an intranet or the Internet. Mobility is increasingly being considered to be a required property of agents [Xu and Wang, 2002], in addition to autonomy, reactivity, pro-activeness, and social ability. Some authors also include mobility as part of a larger set of properties, requiring agents to be simultaneously autonomous, adaptable, knowledgeable, collaborative, persistent, and mobile [Griss and Pour, 2001].

Being adaptable corresponds to having the ability to learn. Being knowledgeable is to have the ability to reason about acquired information. Being collaborative means that the agent can communicate and work collaboratively with other agents, which is related to its social ability. Persistence refers to the ability to retain knowledge over extended periods of time, and across system failures. Being mobile means that an agent is able to interrupt its work in a run-time environment and to resume it in another run-time environment.

The evolution towards mobile agents can be explained by the fact that there has been an ongoing refinement of the concepts surrounding agents and their potential advantages. Originally, agents were regarded as personal assistants [Maes, 1994] whose main purpose would be to provide assistance to a user in certain tasks. For example, agents could be useful to search and organize information from the World Wide Web to be presented to the user, to automatically handle incoming e-mail messages according to their context, or even to choose the best online purchase from a set of several options. For these purposes, an agent would be a static application possibly with a graphical user interface, and with the ability to retrieve and handle information according to its semantic meaning. This kind of agent is called information or Internet agent [Nwana and Ndumu, 1996]. Some authors suggest that in order to design such agents the typical behavior of the human user must be taken into account, so that agents can make optimal decisions or support the user in making those decisions [Sproule and Archer, 2000].

More recently, the concept of agent, which already aimed to mimic some human characteristics such as autonomy and social ability, has been further supplemented with another property: the possibility of being mobile. This ability would allow an agent to transport itself between nodes in a network in order to retrieve information or perform actions locally on those nodes, rather than remotely from another node. Mobile agents can display several advantages when compared to their stationary counterparts or to other approaches [Lange and Oshima, 1999]:

- mobile agents can reduce network load because they can be dispatched to a remote host where interactions will take place locally, without requiring data to be transferred over the network;

- mobile agents can overcome network latency because, by being placed locally at the target host, they can respond in real-time to changes in their environment;
- mobile agents can encapsulate and use proprietary protocols that are different from ordinary remote communication protocols;
- mobile agents can execute asynchronously and autonomously because they can take tasks to remote hosts and execute them locally at those hosts without requiring a continuously open connection across the network;
- mobile agents can adapt dynamically because, in a scenario with multiple mobile agents, they can distribute themselves according to some optimal configuration;
- mobile agents can be used together with heterogeneous systems because they are dependent only on their execution environment and not on platform or transport protocols;
- mobile agents can be robust and fault-tolerant because they have the ability to dynamically react to unexpected circumstances, including the possibility to retreat from a malfunctioning host.

The rationale behind mobile agents is to bring the computation to the data rather than the data to the computation. But despite their advantages, it is clear that mobile agents impose several requirements on the underlying network infrastructure. Above all, the infrastructure must allow application code to be transferred to another host, together with their state, so that agents can resume their execution in another environment. Given the currently available technologies, this is quite a technical challenge, but even if mobility of application code could be achieved, several other issues would arise.

The possibility of having application code moving from host to host introduces some problems [Schoder and Eymann, 2000], most of which are concerned with security issues. In fact, mobile agents require secure execution environments, and the security enforced in these environments must be reciprocal. On one hand, hosts must be protected against dangerous actions that agents may intend to perform, intentionally or not, on the local system. This type of security cannot be successfully implemented by means of ordinary virus scanning since agents exhibit dynamic behavior, and it is impossible to anticipate whether this behavior could be harmful or not. A security solution to protect hosts from agents should be similar to the Java sandbox, which prevents applets from accessing sensitive system software.

On the other hand, an agent must also be protected against hosts which may try to tamper with it, undutifully extract information from it, or manipulate it in some malicious way. Given that mobile agents are supposed to travel around the network until they fulfill their goals, it is expected that agents will remain intact until they complete their tasks. However, hostile systems may interfere with the agent or render it ineffective. Currently, there are no security approaches to protect agents from hosts since, up until now, the most valuable piece is considered to be the host, not an incoming application from a remote host. However, if more and more system functionality is incorporated into mobile agents, as it is expected, then there will be the need to protect these agents as well, since it will become less obvious whether the most sensitive applications reside in the hosts or in the agents that move around the network.

If mobile agents ever turn out to be practical, it is quite likely that they will be subject to severe performance and functional limitations, due to the implementation of security measures. Another

technical disadvantage of mobile agents is that, although they can reduce network load in some situations, they have lower transmission efficiency [Chess et al., 1994] because the full application code must be transferred together with data and state information from one host to another.

5.3.2 Multi-agent systems

The biggest potential of intelligent software agents is not to use agents individually but to let agents work together in what is called a multi-agent system (MAS). A multi-agent system is a collection of agents that interact with each other and with their environments. The purpose of a multi-agent system is to build a network of agents that interact in order to solve problems that are beyond the capabilities or knowledge of each agent [Sycara, 1998b]. The first assumption of multi-agent systems is, thus, that agents are able to interact or communicate with each other. This is the purpose of agent communication languages, which specify the semantics, in addition to the syntax, of a set of language elements for exchanging information between software agents.

Given that a multi-agent system is composed of a set of autonomous agents, there is no global control over the actions of these agents. And because each individual agent can exhibit dynamic behavior, a multi-agent system could end up having an overall unpredictable or chaotic behavior. In order to avoid this situation, the actions of different agents must somehow be coordinated so that the multi-agent system displays a coherent behavior. The second assumption of multi-agent systems, thus, is that the actions of multiple autonomous agents must be coordinated in order to achieve a common goal or the optimal solution for a given problem. This is the focus of agent coordination strategies.

5.3.2.1 Agent communication languages (ACLs)

The widespread adoption and use of object-oriented software development techniques have led to the idea that software applications communicate by calling methods on interfaces of one another. However, the fact that intelligent software agents mimic certain human characteristics suggests that agents could communicate using a language just as humans do, rather than through methods and interfaces. In addition, an intelligent software agent is supposed to have the ability to reason, so the information exchange between agents is referred to as knowledge exchange, in order to distinguish it from data exchange between ordinary software applications. For this reason, agent communication languages specify mainly the semantics, rather the syntax, for a set of language constructs.

The Knowledge Query and Communication Language (KQML) [Finin et al., 1994] is such a language. KQML is a language that supports software agents in identifying, connecting and exchanging information with other agents. This means that KQML specifies how an agent can find other agents, and how it can initiate and maintain exchanges with those agents. For this purpose, KQML introduces a special class of agents called *communication facilitators*. A communication facilitator maintains a registry of existing agents and it routes messages between agents based on their content. The communication facilitator matches agents behaving as information providers to agents behaving as clients, and it can provide mediation and translation services.

The KQML language defines a set of possible interactions between agents and facilitators. These interactions are called *performatives*. KQML performatives allow agents to interact in several ways, such as in a request/reply mode or in a publish/subscribe mode. In addition, some KQML performatives allow facilitators to mediate communication between two agents. In some cases, a facilitator can assist an agent in finding another agent, and then both agents communicate directly. In other cases,

the facilitator can mediate all communication between agents, even after they have already found each other. The KQML specification [Labrou and Finin, 1997] defines a set of thirty six performatives to support all these possibilities of interaction. These performatives are listed in figure 5.31; in some cases, S or R may represent a communication facilitator.

Each KQML performative has a corresponding set of parameters that must be given when the performative is invoked. In many performatives the main parameter is a sentence, which is to be evaluated by another agent. Each agent is assumed to have its own knowledge base containing a set of sentences. The set of sentences in a knowledge base represents the knowledge of the corresponding agent. KQML performatives address mechanisms that allow agents to exchange sentences in different ways. When an agent receives a sentence there are three possibilities. The sentence is true if the agent's knowledge base contains an equal sentence, and it is false if the agent's knowledge base contains a contradictory sentence. If neither the sentence nor a contradictory sentence exists in the knowledge base, the agent may not respond or it may reply with the performative sorry.

One of the main features of KQML is that all performatives are independent of the actual sentences, which can contain any kind of data. Hence, KQML specifies the mechanisms for exchanging knowledge regardless of the way that knowledge is represented. When using KQML, agents can ex-

DISCOURSE PERFORMATIVES	ask-if	S requires a specific R to say whether a given sentence is true or false
	ask-all	S requires all Rs to say whether a given sentence is true or false
	ask-one	S requires any R to say whether a given sentence is true or false
	stream-all	same as ask-all but with multiple responses
	eos	S marks the end of a stream of multiple responses
	tell	S says a given sentence is true
	untell	S does not know whether a given sentence is true or false
	deny	S says a given sentence is false
	insert	S asks R to consider a given sentence as being true
	uninsert	S asks R to reverse the act of a previous insert
	delete-one	S requires R to remove a given sentence from its knowledge base
	delete-all	S requires R to remove all matching sentences from its knowledge base
	undelete	S asks R to reverse the act of a previous delete
	achieve	S asks R to consider a given sentence as being true, and to act accordingly
	unachieve	S asks R to reverse the act of a previous achieve
	advertise	S makes R know that it is able to handle a given type of content
	unadvertise	S makes R know that it is not able to handle a given type of content anymore
	subscribe	S wants to receive updates from R
INTERVENTION AND MECHANISMS OF CONVERSATION PERFORMATIVES	error	S considers R's earlier message to be mal-formed
	sorry	S understands R's message but it cannot provide any information
	standby	S asks R to be ready to respond to a given sentence
	ready	S is ready to respond to a previously received message from R
	next	S wants R's next response to a message previously sent by S
	rest	S wants R's remaining responses to a message previously sent by S
	discard	S does not want any more responses to a previous message
NETWORKING AND FACILITATION PERFORMATIVES	register	S announces its presence to R
	unregister	S asks R to reverse the act of a previous register
	forward	S requests R to forward the message to another agent
	broadcast	S requests R to send a message to all agents
	transport-address	S announces its transport address
	broker-one	S asks R to find one response to a given performative from another agent
	broker-all	S asks R to find all responses to a given performative from other agents
	recommend-one	S wants R to find an agent that may respond to a given performative
	recommend-all	S wants R to find all agents that may respond to a given performative
	recruit-one	S wants R to find a suitable agent to respond to a given performative
	recruit-all	S wants R to find all suitable agents to respond to a given performative

Figure 5.31: KQML performatives between a sender (S) and a receiver (R)

change any content or messages in a language of their own choice, provided that these contents can be encapsulated within a KQML performative. In other cases, such as the last performatives shown in figure 5.31, an agent is also able to pass other performatives as parameters to a facilitator, in order to identify the performatives that other agents implement.

Despite the fact that KQML can be used together with any kind of knowledge representation, there have been efforts to create specific formats for knowledge exchange. Within the field of intelligent software agents, one of the most important efforts towards this goal has been the Knowledge Interchange Format (KIF) [Finin et al., 1995]. KIF can be used to express simple data, such as numeric quantities and character strings, and the relationships between those data. KIF includes logical operators and supports user-defined operators as well. With these operators it is possible to define concrete relationships (e.g. "A is greater than B") or abstract relationships (e.g. "A is interested in B"), respectively. An agent can build a knowledge base by collecting several KIF sentences, and it can use KIF sentences as parameters in KQML performatives.

Another important agent communication language is FIPA ACL [FIPA, 2000b]. With respect to its underlying principles, the FIPA ACL language is almost identical to KQML. With FIPA ACL, as with KQML, agents communicate by means of performatives. These performatives are sent from one agent to another according to the message structure shown in figure 5.32. However, FIPA ACL has different performatives from those of KQML. In addition, FIPA ACL does not make use of the concept of communication facilitators. As a result, KQML and FIPA ACL, though having a similar syntax, have quite different semantics. Another difference is that, whereas KQML is independent of the knowledge representation format, FIPA ACL requires agents to have some understanding of a specific content language called FIPA SL [FIPA, 2000c].

5.3.2.2 Agent coordination

Besides having the ability to communicate, agents must be able to coordinate their actions with each other so that a community of multiple agents exhibits coherent behavior. The main reasons that explain the need for coordination in a multi-agent system can be summarized as follows [Nwana et al., 1996]:

- to prevent anarchy and chaos, since each agent is autonomous and it reasons based on a limited view of reality, which may be different for each agent;

COMMUNICATIVE ACT	performative	Type of communicative act of the ACL message
PARTICIPANTS	sender	Identity of sender of the message
	receiver	Identity of the intended recipients for the message
	reply-to	Identity of the agent to which replies are to be sent
CONTENT OF MESSAGE	content	Content of the message according to the current performative
DESCRIPTION OF CONTENT	language	The language in which the content is expressed
	encoding	The encoding of the message content
	ontology	The ontology that gives meaning to the symbols in the message content
CONTROL OF CONVERSATION	protocol	The interaction protocol the sending agent is employing with this message
	conversation-id	Identity of this communicative act within a sequence of communicative acts
	reply-with	An expression that must be used in the response to this message
	in-reply-to	An expression that references an earlier message
	reply-by	The latest time by which the sending agent will receive a reply

Figure 5.32: FIPA ACL message structure

- to meet global constraints such as, for example, a pre-specified budget for the sum of work of all agents;
- to distribute expertise, so that agents can rely on the knowledge of each other instead of having to possess all necessary knowledge for each task;
- to enforce dependencies between actions whenever those dependencies apply, such as when the action of an agent must precede the action of another;
- to improve efficiency, for example by sharing information between agents without requiring each agent to obtain that information from the original source.

Four main approaches to achieve coordination in multi-agent systems have been identified [Nwana et al., 1996]. The first approach is to devise techniques based on organizational structure. The hierarchical relationships between responsibilities within an organization can be used as a model to coordinate agents, where master agents command the actions of slave agents. In this case, even if the slave agents do not communicate with each other, they will ultimately report their results back to a common master agent. A second approach is to allow agents to establish contracts so that they commit themselves to perform certain tasks without requiring hierarchical control. The third approach is to engage agents in devising common plans, with or without the help of a coordinating agent. In this approach, it may be necessary to solve conflicts between the original plans of different agents.

This is, in fact, the main focus of a fourth approach that is exclusively based on negotiation between agents. In general, negotiation begins with one agent making a proposal and then another agent evaluates and checks that proposals against their own preferences. This will possibly produce a counter-proposal, which will be sent to the first agent. Now the first agent will evaluate the counter-proposal and, if needed, will produce another proposal. This process is repeated until both agents find an acceptable proposal, i.e., one that satisfies their preferences totally or partially.

There are several research fields that may contribute to the development of coordination techniques for multi-agent systems. For example, classic work in organization structuring has identified four coordination mechanisms used in organizations, besides simple hierarchical control [Tolksdorf, 2000]. In addition, game theory and artificial intelligence techniques can be useful in negotiation-based coordination approaches. An especially interesting idea is that workflow management systems could be employed to coordinate the actions of intelligent software agents. The relationship between workflow management and intelligent software agents is discussed ahead in section 5.3.6.

One of the main fields that could have an influence on the design of coordination techniques for multi-agent systems is coordination theory [Malone and Crowston, 1994]. Research efforts in this area have led to the development of coordination models and languages [Papadopoulos and Arbab, 1998]. One of such languages is STL++ [Schumacher et al., 1999][79]. The STL++ language has basically five constructs: *blops*, *agents*, *ports*, *events* and *connections*. Each agent is assumed to have one or more ports, and these ports may or may not be linked to other agents by means of connections. Using a connection between ports, an agent can communicate events to another agent. A blop represents a common environment for a group of agents, where communication between those agents takes place. Each blop may also have one or more ports in order to allow communication between agents belonging to different blops. This situation is illustrated in figure 5.33.

[79]The author has personally met M. Schumacher, one of the authors of STL++, at the ICEIS'99 conference in Setúbal, Portugal, in March 1999

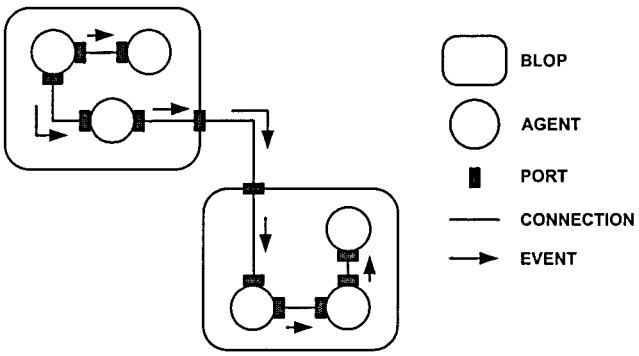


Figure 5.33: Elements of the STL++ coordination language

One interesting feature is that STL++ defines four different types of ports: *blackboard ports* and *group ports* for communication between several agents, and two kinds of *stream ports* for direct, one-to-one communication between agents. The difference between blackboard ports and group ports is that a blackboard port works like a message repository where messages are kept persistently, and from which agents may retrieve messages at any time. The group port, on the other hand, dispatches messages immediately to all recipients without storing them.

Of special interest to this work is the fact that the STL++ language can be used to express the structure and the sequence of events in a inter-enterprise trading system [Schumacher et al., 1999]. Figure 5.34 depicts a hypothetical trading system built as a multi-agent system. Each enterprise creates queries to buy or sell products, and submits those queries to the trading system. Within the trading system there is one broker agent for each enterprise, and those queries are transmitted to corresponding broker agent via stream ports.

When a broker agent receives a query, it publishes the query within the trading system by means of a blackboard port. This blackboard port is represented as the trade unit blackboard in figure 5.34. The trade manager continuously monitors the trade unit blackboard in order to find matches between queries from different enterprises. Whenever a match is found, the trade manager instructs the relevant broker agents to assist the corresponding enterprises in performing the desired business exchanges. The broker agents will then establish a stream port for one-to-one communication, and the enterprises will be connected to each other through this link.

The STL++ coordination language is a valuable help in describing the interactions and dependencies between agents in a multi-agent system. However, the challenge of coordinating multiple agents must be met with appropriate coordination techniques, which may be sensitive to several issues such as the number of agents or the dynamics of their environment. For example, some coordination strategies may work well for a few number of agents, but they may be ineffective to deal with hundreds of agents. Ultimately, there are three main dimensions that may affect a coordination strategy [Durfee, 2001]: the agent population, the environment, and the solution. The first dimension concerns the quantity, heterogeneity and complexity of agents. The second dimension includes those issues related to resource sharing between agents, environment dynamics, and distributivity of the environment. The third dimension, the solution, should be evaluated based on its computational requirements, its QoS characteristics, and its ability to cope with unexpected deviations.

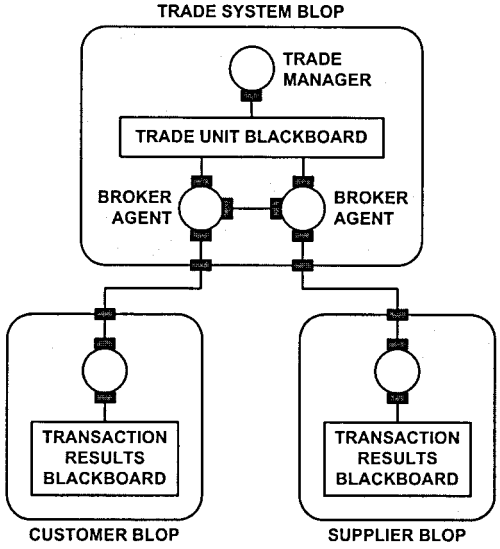


Figure 5.34: A trading system represented with STL++[80]

5.3.3 Standardization efforts

The developing concept of multi-agent systems has forced researchers to provide solutions for agent communication and coordination. If these were the only problems to tackle, then some of the above approaches would be enough to devise and implement a multi-agent system. But this is not the case. Multi-agent systems bring a myriad of challenges that must be addressed. How to identify agents, how to locate agents, how to implement secure execution environments, how to move agents between environments, and how to interface agents with humans are some of the challenges associated with multi-agent systems.

Every multi-agent system must cope with these requirements. But if each agent system devises its own approach to meet these challenges, then agent systems will, in general, be unable to interoperate with each other. Thus, the field of multi-agent systems needs standards in order to ensure that different agent systems will be interoperable. Without restricting the development approaches for multi-agent systems, interoperability can be achieved by defining standard interfaces that those systems should implement. This is precisely the approach of existing standardization efforts in the field of software agents.

Besides interoperability, there are additional reasons for the need of standards in software agent systems [O'Brien and Nicol, 2001]. Agents need commonly agreed mechanisms to communicate with each other in order to exchange knowledge, negotiate for services, and deliver tasks. Agents need facilities to identify each other, to locate each other and, whenever applicable, to move from one platform to another. Agents also need a secure and trusted environment where they can operate and exchange sensitive information. Finally, agents require means of accessing legacy systems and of interacting with human users. The currently available standards address some of these issues.

[80][Schumacher et al., 1999]

5.3.3.1 The FIPA specifications

The FIPA standard [Dale and Mamdani, 2001] is a comprehensive set of specifications ranging several areas of concern for agent-based systems. First, there is an abstract architecture specification [FIPA, 2000a], which identifies fundamental elements of agent systems. These elements include agents and agent communication languages, performatives and their content, directory services where agents advertise their presence, messages between agents, message encoding syntax, the vocabulary shared by a community of agents (i.e., the *ontology* [Uschold and Gruninger, 1996]), and message transport properties. The abstract architecture specification describes the relationships between these elements in an generic way, regardless of any particular implementation approach or technology.

As suggested in figure 5.35, the elements defined in the abstract architecture specification (a) are the common ground for the remaining four areas addressed by FIPA standards. The agent message transport specifications (b) deal with the representation and delivery of messages across different network transport protocols including, but not limited to, HTTP and WAP. The agent communication specifications (c) comprise the FIPA ACL communication language and the FIPA SL content language, as previously discussed. They have exactly the same purpose as KQML and KIF, respectively. The agent management specification (d) describes directory services for agent naming and location, as well as the management functions to control the agent life cycle. The agent-based application specifications (e) describe how FIPA standards can be employed in several application scenarios such as personal assistance, multimedia broadcasting, and network management.

5.3.3.2 The MASIF facility

The OMG's Mobile Agent System Interoperability Facility (MASIF) [Milojicic et al., 1998] is another standardization initiative that addresses some of those same areas, but focusing particularly on mobile agents. The MASIF facility, which is built on the CORBA architecture and several of its services, specifies a set of interoperable, CORBA-based interfaces for mobile agent systems. The MASIF facility divides the functionality of a mobile agent system according to two main interfaces: the MAFAgentSystem interface and the MAFFinder interface. The MAFAgentSystem interface supports agent management and agent transfer. The MAFFinder interface allows an agent system to maintain information about the name and location of agents.

The MAFAgentSystem interface supports agent management by defining the methods to suspend, resume, and terminate an agent's execution. Hence, it allows an agent system to control the agents in another agent system. The MAFAgentSystem interface also supports agent transfer because it defines methods for an agent system to receive agents from another agent system, and to fetch their classes in order to create local agent instances. The MAFFinder interface, on the other hand, defines methods

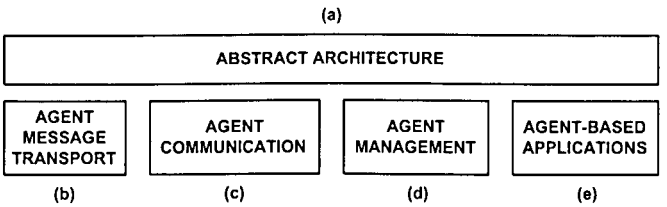


Figure 5.35: Areas addressed by FIPA standards

that allow an agent system to register the name and location for each agent, and allows other agent systems to retrieve that information.

The MASIF facility does not address agent communication languages, but it does consider several security requirements for mobile agent systems. A basic requirement is that agents must be authenticated when they are transferred to another agent system, preferably by carrying their credentials as they move from one agent system to another. These credentials may not refer to the agent itself, but to the human user (the principal) that the agent acts on behalf of. In addition, an agent system should provide its own credentials whenever it requests the creation of an agent instance on another agent system - this is referred to as client authentication. The MASIF facility realizes that server authentication is also imperative, so it requires mutual authentication of agent systems whenever they interact.

In order to implement such security measures, as well as other features, the MASIF facility relies on several standard CORBA services. For example the Naming Service, which binds names to objects, can be useful for implementing MAFFinder-related functionality. The Life Cycle Service can be used to create and terminate agent instances, and to suspend and resume their execution, which is one of the purposes of the MAFAgentSystem interface. The Externalization Service can assist in recording an agent's structure and state in order to move that agent from one system to another. Finally, the Security Service can be used to authenticate agents, and to mutually authenticate agent systems. However, the OMG realizes that the MASIF facility, as well as mobile agent technology in general, introduces security requirements that are beyond the current CORBA security capabilities [OMG, 1998].

5.3.4 Agent platforms

Given that the design and implementation of agent-based systems introduces demanding challenges, it is impossible to consider the design of an agent-based system without paying attention to how other authors have dealt with those challenges. The FIPA and OMG standards are, in part, a collection of contributions from several authors which have faced some of the most important issues that pertain to the design of agent-based systems. However, these standards do not present concrete approaches to tackle with agent system requirements. Instead, they present an abstract view on the concepts, structure and approaches that are common to many agent systems. This is indeed the purpose of standardization initiatives in general.

Yet, the availability of standards does not imply that the design of agent systems has become easier. The standards specify mainly the external interfaces for those systems in order to ensure that different agent systems will be interoperable. The internal architecture of an agent system and the agent environment, as well as the mechanisms that implement agent properties and behavior, must be dealt with on a case basis. This situation hampers the development of agent-based systems because every potential agent application will eventually face the need to implement the same infrastructural capabilities. As a result, the design of agent-based systems must focus on these common capabilities instead of focusing exclusively on the details of the particular application.

In this regard it should be noted that, to some extent, the same situation applies to workflow management systems. The WfMC standards specify the external interfaces that workflow management systems should implement, but each workflow management system is free to have its own internal architecture, which is somewhat different from workflow system to workflow system. The present situation regarding workflow management systems is that, whenever a possible application for such systems is found, an entire compound of workflow functionality must usually be devised from scratch.

As a consequence, the development of each workflow solution is plagued with concerns about internal system details, rather than focusing exclusively on the details that pertain to the scenario where workflow management will be employed.

In the case of multi-agent systems there is a way to avoid this situation: there are agent platforms. An agent platform implements basic supporting functionality for multi-agent systems, and it allows agent systems to be built on top of that functionality. The last few years have witnessed the development of several agent platforms, as more and more efforts were devoted to the development of generic agent platforms. Currently, there are many agent platforms available with different characteristics [Mangina, 2002]. Some of these platforms are available as commercial products, whereas others are available as open research projects, as shown in figure 5.36.

Given the availability of these agent platforms, multi-agent systems do not have to be built from scratch anymore; they can rely on the facilities provided by agent platforms. One of the consequences, however, is that some system characteristics will not depend on the agent-system itself, but on the underlying agent platform. For example, agent mobility will depend on the mechanisms the agent platform provides to record an agent's execution state, transfer it to another environment, and resume its operation there. These characteristics will have a strong impact on the system's interoperability. Ideally, the underlying platform should comply with agent-related standards, otherwise

Agent Platform	URL	Supplier	Availability
ADK	http://www.tryllian.com/adk.html	Tryllian	Commercial (Licensed)
AgentBuilder	http://www.agentbuilder.com/	IntelliOne Technologies	Commercial (Licensed)
AgentSheets	http://www.agentsheets.com/	AgentSheets Inc.	Commercial (Licensed)
AgentTool	http://en.afit.af.mil/ai/agentool.htm	Kansas State University	Academic (Free)
Aglets	http://www.trl.ibm.com/aglets/index_e.htm	IBM Japan	Commercial (Free)
Bee-gent	http://www2.toshiba.co.jp/beegent/index.htm	Toshiba Corporation	Commercial (Free)
CABLE	http://public.logica.com/~grace/Architecture/Cable/public/	Logica UK Ltd	Commercial (Licensed)
Comet Way Agent Kernel	http://www.cometway.com/	Comet Way Ltd.	Commercial (Free)
CORMAS	http://cormas.cirad.fr/indexeng.htm	CIRAD	Commercial (Free)
Cougaar	http://www.cougaar.org/	DARPA / BBN	Other (Free)
Cybele	http://www.opencybele.org/	Intelligent Automation Inc.	Commercial (Free)
D'Agents	http://agent.cs.dartmouth.edu/	Dartmouth College	Academic (Free)
DECAF	http://www.eecis.udel.edu/~decaf/	University of Delaware	Academic (Free)
FIPA-OS	http://fipa-os.sourceforge.net/index.htm	Emorphia Ltd.	Commercial (Free)
Grasshopper	http://www.grasshopper.de/	IKV++ Technologies AG	Commercial (Licensed)
JACK	http://www.agent-software.co.uk/	Agent Oriented Software Pty. Ltd	Commercial (Licensed)
JADE	http://sharon.cselt.it/projects/jade/	TILAB	Commercial (Free)
JADE / LEAP	http://leap.crm-paris.com/	LEAP Consortium	Commercial (Free)
JAFMAS / JIVE	http://www.ececs.uc.edu/~abaker/JAFMAS/	University of Cincinnati	Academic (Licensed)
JATLiteBean	http://kmi.open.ac.uk/people/emanuela/JATLiteBean/	University of Otago	Academic (Free)
JESS	http://herzberg.ca.sandia.gov/jess/	Sandia National Laboratories	Academic (Free)
J-SEAL2	http://www.jseal2.com/	CoCo Software	Commercial (Licensed)
Kaariboga	http://www.projectory.de/kaariboga/index.html	Other	Other (Free)
Living Markets	http://www.living-systems.com/	Living Systems AG	Academic (Free)
MAML	http://www.maml.hu/maml/initiative/	Agent-Lab Ltd.	Commercial (Licensed)
MASSYVE KIT	http://www.gsigma-grucon.ufsc.br/massyve/mkit.htm	Federal University of Santa Catarina	Academic (Free)
MonJa	http://www.melco.co.jp/rd_home/java/monja/	Mitsubishi Japan	Commercial (Free)
RETSINA	http://www-2.cs.cmu.edu/~softagents/	Carnegie Mellon University	Academic (Licensed)
SIM_AGENT	http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html	University of Birbingham	Academic (Free)
StarLogo	http://www.media.mit.edu/starlogo/	MIT MEDIA lab	Academic (Free)
TuCSon	http://www.lia.deis.unibo.it/Research/TuCSoN/	DEIS Universita di Bologna	Academic (Licensed)
Voyager	http://www.objectspace.com/products/voyager/	Recursion Software, Inc.	Commercial (Licensed)
ZEUS	http://www.labs.bt.com/projects/agents/zeus/index.htm	BTexact Technologies	Commercial (Free)

Figure 5.36: Some of the currently available agent platforms[81]

[81]This list has been compiled from a similar table [Mangina, 2002] and from additional entries inserted by the author.

interoperability will be compromised. In practice, though, many of the agent platforms presented in figure 5.36 are not compliant with any agent-related standard.

There is a explanation for this: once again, this is due to the complexity of the requirements introduced by multi-agent systems. In many cases, for example with IBM's Aglets [Lange and Oshima, 1998], crucial functionality such as agent registration, mobility and data exchange is implemented using features of a particular technology (e.g. Java). Because these features are specific to the chosen technology, it is not straightforward to implement external standard interfaces such as those defined in MASIF. In other words, many agent platforms cope with agent system requirements by employing techniques that may not be interoperable with other agent-based systems. Fortunately, there are few exceptions to this trend: FIPA-OS and Grasshopper, for example, have been purposely built as standard-compliant agent platforms.

The FIPA-OS Toolkit, as its name suggests, is a FIPA-compliant agent platform. Essentially, it is a comprehensive set of components implemented with Java technology, and its purpose is to support the rapid development of agent-based applications. The platform includes a base class, called GenericAgent, which is to be extended with the intended agent functionality. The advantage of using this base class is that it already implements several specific features such as, for example, agent registration. Incidentally, the FIPA-OS platform includes some agents - the name service agent and the directory service agent - that support agent registration and discovery.

The FIPA-OS toolkit has a layered architecture with several components, as shown in figure 5.37. In a few words it can be said that, at the lowest level, the Message Transport Protocols implement basic mechanisms for exchanging messages over the network. FIPA-OS provides two implementations of the Message Transport Protocols, one based on RMI and another on IIOP. At the next upper level, the Message Transport Service provides the ability to exchange messages with other agent implementations. The Conversation Manager ensures that agents exchange messages according to a given performative. The Task Manager allows agent execution to be divided into separate, independent tasks that can be run concurrently, and the Conversation Manager will keep track of the conversation state within each task. The Agent Shell is able to load and initialize agents, and to start any other components that an agent may require.

As for Grasshopper, it is an agent platform that was developed with the determinate goal of being the first MASIF-compliant agent platform. Grasshopper makes use of the same concepts and approach that the MASIF facility introduced. The Grasshopper platform provides agent registries that are accessible through the MAFFinder interface, and it provides an agent execution environment that implements the MAFAgentSystem interface. Grasshopper, like most agent platforms, is implemented with Java technology. The reason for this is that platform-independence is a desirable feature of any agent platform, since it is a crucial factor in order to enable agent mobility across different host

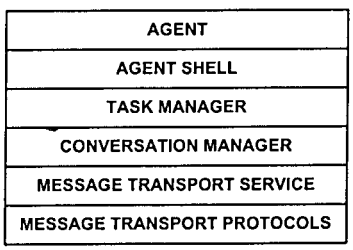


Figure 5.37: Layers of the FIPA-OS architecture

machines. A noteworthy exception is the D'Agents platform, which is a modified version of the TCL language, and which is not straightforward to use with hosts other than UNIX-like machines.

In a similar way to the MASIF facility, so too does Grasshopper include a set of additional services in its agent execution environment. The Communication Service is responsible for all interactions between components within the Grasshopper platform. These interactions may take place via RMI, IIOP or TCP/IP socket connections. The Registration Service maintains information about currently hosted agents. The Management Service allows the monitoring and control of agents, such creating and removing them from the platform, as well as suspending and resuming their execution. The Transport Service supports the migration of agents from one execution environment to another. Lastly, the Security Service distinguishes between *external security* and *internal security*.

External security protects the interactions between components from external threats. For example, agents use digital certificates and SSL when they communicate with registries. On the other hand, internal security protects components from unauthorized access, i.e., it may prevent one agent to interact with another. Internal security is achieved by authenticating the user that an agent acts on behalf of.

The Grasshopper platform, which was initially developed with the aim of being MASIF-compliant, is now FIPA-compliant as well. In fact, the MASIF- and FIPA-compliant functionality are supplied as two separate extensions to the main platform functionality. A third, recent extension, which is still under development, is Webhopper. Built on servlet technology, Webhopper provides the ability to control and interact with agents through the Web. The Grasshopper platform thus comprises the Grasshopper Core System, the MASIF Add-on, the FIPA Add-on, and the Webhopper Extension.

5.3.5 Applications of software agents

The improvement of agent coordination and communication techniques, the FIPA and OMG standards, and the availability of increasingly mature agent platforms are the main elements that guide the development of agent-based systems. As it becomes easier to develop agent-based systems, since many of their requirements are being addressed by extensive research, standardization bodies and existing agent platforms, software agents find an increasing number of applications. The following paragraphs describe some applications of agent-based systems in several areas related to this work.

5.3.5.1 B2C e-commerce

Agent-based systems are especially advantageous to the practice of e-commerce. In B2C e-commerce, agent-based systems can assist the human user in locating and retrieving relevant information from the Web, and in organizing that information in such a way that it becomes easier for the user to evaluate and compare several purchase options [Brenner et al., 1998]. In this case, an agent has a similar purpose to a personal assistant, which is provided with user preferences and is able to estimate the interest of any piece of information based on those preferences. This allows the user to cope with overwhelming amounts of information available on the Web, and to efficiently select the relevant information, which could otherwise take an unacceptably long time to accomplish.

An example of such an agent-based system is Data Agents [Martins et al., 2001], which finds the online stores that sell the desired product and lists the prices for the products found. Basically, this is achieved in three steps. First, the user specifies the desired product and its characteristics. Then, a purchase agent will search for the product in a set of servers previously registered in the system. The purchase agent, which is a mobile agent, visits each server and retrieves product information, if

available, from that server. After collecting the results, the Data Agents system presents them to the user, either via a graphical user interface or via e-mail. The Data Agents system has been implemented with Aglets.

More than collecting information, agent-based systems can even foster a sense of community among their human users. This is demonstrated by the Personal Agent Framework (PAF) [Case et al., 2001], which contains a set of agents devoted to personal assistance. One of these agents is the Profile Manager: it stores information about user interests. These interests may be publicly known, or they may be kept private. Based on this information, a second agent (called Bugle) generates a daily newsletter containing articles that are of interest for the user. A third agent (Gravepine) periodically notifies the user via e-mail about other users who have similar interests. The iVine agent allows users to locate other users with common interests. The Pandora agent suggests new interests for the user to consider. The Radar agent locates relevant information sources every time the user reads or prepares information to submit to others. The Jasper agent determines who receives which materials. The ProSearch agent continuously searches for new interesting materials, and the Prosum agent is able to summarize large documents.

5.3.5.2 B2B e-commerce

The ability of agents to replace humans in searching and sorting information from the Web can be advantageous to B2B e-commerce as well. In B2B e-commerce, agents can replace buying organizations in collecting information from potential suppliers, in negotiating contract conditions, and in selecting the best supplier. The ABROSE project [Einsiedler et al., 1999], whose purpose is to design and implement an agent-based system supporting brokering in an electronic marketplace, is an example of such an approach. According to ABROSE, brokering is defined as the search and matching of buyers and sellers. These search capabilities are symmetric: for a given product, a buyer may search for an appropriate seller, and a seller may search for potentially interested buyers. The ABROSE system is divided into two different domains: the user domain, i.e. the environment for buyers and sellers, and the broker domain, which is the environment for software agents that implement the electronic marketplace's brokering capabilities.

In the broker domain, there are two kinds of agents: Transaction Agents (TAs) and Mediation Agents (MAs). A Transaction Agent represents a single enterprise in the electronic marketplace and, despite belonging to the broker domain, it is controlled by that enterprise. Essentially, the Transaction Agent is the interface between the enterprise that it represents and the electronic marketplace. On the other hand, the brokering capabilities are implemented with Mediation Agents, which are able to communicate with all Transaction Agents in order to match buyers with suppliers. Mediation Agents also monitor the actions of Transaction Agents and report those actions to the broker domain's management in order to provide a global view on the operation of the electronic marketplace.

Within the broker domain, all these agents communicate by means of an agent communication language based on four performatives only. Two of these performatives are used to express system failure and malformed messages, respectively. A third performative allows an agent to submit queries to other agents. An important feature of ABROSE is that each agent may forward a query to another agent, if it does not have an answer to the original query. This may happen when a Transaction Agents forwards a query to a Mediation Agent, or when a Mediation Agent forwards a query to another Mediation Agent. The fourth performative can be used either as a response, when an agent replies to an earlier query, or as a notification, whenever a Transaction Agent reports one of its actions to a Mediation Agent.

The Magnet system [Collins et al., 2001] employs a similar approach, but uses information about the performance of suppliers. According to the Magnet system, each partner has its own agent. The customer agent issues a Request for Quote (RFQ) concerning a given product and, in reply to this RFQ, supplier agents submit their own bids. The customer agent will then award one of these bids, and it will initiate exchanges with the corresponding supplier. The Magnet system has a central node, called the Market, which connects customer and supplier agents, and mediates all interactions between those agents.

The interesting characteristic of the Magnet system is that an RFQ identifies a set of interdependent tasks to be performed by one or more suppliers. The customer agent is also given a schedule that specifies the maximum duration for each of those tasks. When a supplier agent submits a bid in reply to a RFQ, the bid contains the price and expected duration regarding those tasks that the supplier is able to perform. Other tasks may have to be performed by different suppliers. Notwithstanding, the customer agent knows that each supplier may fail to complete a task within the desired time period. Its goal is therefore to select the best supplier for each task in order to minimize deviations from the initial schedule. This is possible because the Magnet system, which mediates all interactions between customer and supplier agents, maintains up-to-date values of performance indicators (successes and delays) for each supplier and task type. With this data, the customer agent employs sophisticated algorithms for bid selection, such as simulated annealing.

Another approach is to consider several other parameters besides performance indicators, and to treat automated negotiation as a distributed constraint satisfaction problem (DCSP) [Merlat, 1999]. In this case, the goal is to negotiate the provision of one or more external services, each of which is described by a set of tuples. Each parameter in a service tuple may create a dependency of one service on another service, in a similar way to the interdependence between tasks in the Magnet system. For example, in a travel agency application scenario there are two services: the flight service and the hotel service. These services are related because, if the flight service requires a overnight stop at some location, then the hotel service must book a hotel room in a nearby place for that night.

In a DCSP, the customer requirements are expressed as a set of constraints. In practice, this means that some of the service parameters are fixed (e.g. the desired destination), whereas other parameters can be slightly adjusted (e.g. time of arrival). These adjustable parameters are referred to as constraint variables. In agent-based systems using DCSP algorithms, each agent is responsible for one constraint variable. Furthermore, agents communicate with each other in order to select variable values that are mutually compatible. The MOTIV project [Merlat, 1999] has implemented such an agent-based system using the Voyager platform, and has demonstrated its usage precisely in the case of a travel agency application.

5.3.5.3 Manufacturing and Enterprise Integration

Besides emerging e-commerce practices, the possibilities of agent-based systems can have an impact in traditional fields too, such as manufacturing and Enterprise Integration. Agent-based systems have five characteristics of special importance for modern manufacturing: they are modular, decentralized, changeable, ill-structured, and complex [Parunak, 1998a]. Manufacturing approaches can benefit from these characteristics because [Parunak, 1998a]:

- any industrial entity has its own set of state variables, and the interface with its environment can be clearly defined (modularity);

- given the proliferation of different manufacturing devices across increasingly larger factories, control and scheduling problems must be considered from a non-hierarchical point of view (decentralization);
- the modern trend towards increased product variety and shorter product life cycle makes the manufacturing environment subject to frequent change (changeability);
- in dynamic manufacturing environments not all of the necessary structural information is available when the manufacturing system is designed, instead it is discovered when the system operates (ill-structure);
- increasing product complexity and variety means that a larger number of manufacturing equipment must be scheduled and controlled (complexity).

These requirements have motivated the introduction of agent-based approaches in the design, simulation, scheduling and control of manufacturing systems, and numerous research projects have been undertaken in order to explore the advantages of this new paradigm [Parunak, 1998b]. The different agent-based architectures that have arisen since then can be classified into three main categories [Shen and Norrie, 1999a]. The first category is that of agent-based manufacturing systems which have tried to mimic some of the hierarchical relationships between business units and resources: this is called the hierarchical approach. The second category, the federated approach, usually leads to architectures that contain broker agents, which mediate interactions between agents that perform specific tasks. The third category is the autonomous approach, which encapsulates resources and engineering tools as intelligent agents, and provides those capabilities as special services.

One example of the autonomous approach is the AARIA project [Baker, 1999]. The AARIA project includes a set of three components that are illustrative of the issues involved in designing an agent-based system. First, AARIA has developed an agent architecture that represents the manufacturing system according to its information modularity and its physical reality. Second, AARIA has developed a simulation tool that allows agents to reason about their actions before performing them. And third, AARIA has devised an infrastructure to support the implementation of those agents.

The AARIA agent architecture identifies three different kinds of agents. Resource broker agents encapsulate resources such as people, machines and facilities, which have a limited production capacity. Part broker agents are responsible for material handling and inventory management. Unit-process broker agents combine the actions of resource broker agents and part broker agents in order to make other parts and products. According to the AARIA architecture, these three kinds of agents negotiate with each other on the grounds of price and delivery time. Agents should also able to modify their plans, when faced with machine failure or emergency jobs, and to make an efficient use of resources in order to minimize the cost of each action. The AARIA agent infrastructure has been implemented on top of the Cybele agent platform (listed in table 5.36).

The AARIA simulation tool relies on the ability of agents to run in a simulation mode, without any changes to physical entities (resources and materials). Thus, from a customer request, the simulation tool is able to portray the requests for bids that propagate down to part broker and unit-process broker agents, and it is able to simulate the behavior of the system if the agents would be committed to those tasks. The AARIA project has also developed a scheduling algorithm, which uses both forward and backward scheduling techniques[82].

[82]Forward scheduling is the scheduling of future manufacturing activities at the time when production starts. Backward scheduling refers to the same activities, but schedules them backwards from a due date, so that the product will be ready by that date.

Another simulation tool for agent-based manufacturing systems is the MABES emulation system [Ivezic et al., 1999]. A distinctive feature of MABES, however, is that it places a strong focus on manufacturing processes, and it assumes that each agent is responsible for monitoring and acting on a component of a manufacturing process. Hence, MABES identifies these four types of agents: the customer agent, the process center agent, the process agent, and the stack agent. A process center agent manages a single production stage, in which one or more process agents perform manufacturing tasks. The stack agents hold parts and materials between production stages. These agents do not overlap in their competencies, and together they cover the whole manufacturing process. The MABES tool can analyze alternative scheduling strategies for this process, it presents an animated view on message passing between agents, and it can display manufacturing process statistics.

These and other research projects suggest that agent-based approaches provide new insight into how manufacturing systems can be designed and operated. But the design of a manufacturing system may be part of a larger enterprise integration endeavor, so the possibilities of employing agents within the wider scope of enterprise integration architectures has been investigated as well. One possibility is to build a multi-agent system on top of an existing enterprise integration infrastructure. Such is the case of the Product Realization Environment (PRE) [Pancerella and Berry, 1999], which has been enhanced with agent functionality.

The PRE is an infrastructure for tool integration, data exchange, and interoperability in an enterprise-wide environment. This infrastructure has been enhanced with a multi-agent system that supports four types of agents. User agents implement the interface between human users and the remaining agents. Resource agents interact with the available resources and manage their usage. Information agents reason about information sources and create relationships between data in enterprise resources. Broker agents gather resources and data sources across the enterprise, and provide those capabilities as special services to internal or even external entities.

The multi-agent system built for PRE illustrates how agents can maintain and use knowledge about the enterprise. For example, when the user is looking for a specific kind of software application, the user agent will attempt to apply previous knowledge based on prior requests for that resource type and, if necessary, it will query the resource agents. The resource agents may use the PRE environment or application-specific capabilities to retrieve a list of similar applications. The user agent will then reason over both new and previous knowledge, and it will generate a response to the user.

Another example of an enterprise integration infrastructure that has been enhanced with agent functionality can be found in the InfoSleuth project [Woelk et al., 1995], which extends the Carnot project [Huhns et al., 1992]. On top of the Carnot infrastructure, the InfoSleuth project has devised an agent-based architecture that simplifies the development and maintenance of distributed applications. For this purpose, the InfoSleuth project has implemented two kinds of agents: procedurally scripted agents and declarative agents. Procedurally scripted agents are essentially scripts that are able to communicate synchronously and asynchronously in order to interfere with the execution of each other. Declarative agents are more akin to intelligent software agents since they have reasoning capabilities and they communicate via KQML performatives. The InfoSleuth project uses declarative agents to discover data patterns and relationships across databases within the enterprise, including text, audio, image, and video databases.

The SBD program [Dabke, 1999] has explored a different approach from the PRE environment and the InfloSeuth project: instead of relying on an existing enterprise integration infrastructure, the SBD program has developed its own infrastructure and it has implemented several agents on top of this infrastructure. The SBD infrastructure, called the Core Processing System, is an object-oriented infrastructure built on CORBA, and it is composed of three layers. The bottom layer is a collection

of basic services such as naming, security and persistence. The middle layer comprises a set of collaboration services provided by several information agents. The top layer is the user interface to the system that supports both CORBA client applications and access via the Web.

The SBD Core Processing System includes five information agents. The Query Agent is the logical repository for all objects in the SBD environment, and its functionality is similar to that of an object-oriented database. The Workflow Agent, which makes use of a proprietary process modeling language called SWL, supports tasks assigned to human users as well as operations on CORBA and Java objects. The NetBuilder Agent is a tool for expressing data and control dependencies between objects. The Mediation Agent allows the user to define data transformations between tasks. The Notification Agent implements a publish-subscribe messaging model for communication between agents, users, and applications.

All of the SBD information agents share the same internal architecture. Basically, each of these information agents can be regarded as implementing a different process model, and they contains an inference engine that automates its execution. For example, the Notification Agent has a model that defines what each subscriber wants to be done when a certain message is published, so that every time a message is received, the Notification Agent must run its model in order to perform the required actions. This model is called the Agent Information Management Model, and it is one of main the components of an information agent's internal architecture.

Besides the Agent Information Management Model, information agents have a Dynamic Invocation Adapter, which allows them to invoke remote operations on another object without having known those operations before. The only requirement is that the object to be invoked must be registered in the naming service. The third component of an information agent's internal architecture is the CORBA wrapper, which exposes the functionality of that agent to the remaining objects in the SBD infrastructure. Other agents will then be able to communicate with this agent, as long as they specify some elements, namely the message context (sender and message identifier), its content, and its intent (whether it is a query, assertion or task to be performed).

5.3.5.4 Supply Chain Management and Virtual Enterprises

The SBD program argues that its infrastructure is the kind of collaboration infrastructure required by virtual enterprises[83]. In fact, some authors have emphasized the need for collaborative agents in order to implement supply chain management and virtual enterprises [Shen and Norrie, 1999b]. In particular, these authors propose an Infrastructure for Collaborative Agent Systems (ICAS), which comprises domain-specific collaborative agents, domain-independent components (such as knowledge management tools and ontology servers), and common collaboration services. Such an agent system would bring to supply chain management and virtual enterprises similar advantages to those identified in the case of manufacturing systems.

The MetaMorph II project [Shen and Norrie, 1998] has devised an agent-based architecture for supply chain management and, in this architecture, mediator agents play a central role. Within a single enterprise, all manufacturing resources and participants are represented by their own mediator agent. Then, an enterprise mediator connects all these mediator agents together, an it connects to the mediator agents of other enterprises, such as external suppliers and distributors. Despite this hierarchical arrangement of mediator agents, there is no hierarchical control and, in general, any mediator agent can communicate with any other mediator agent even if both agents belong to different

[83] In addition, the SBD program realizes that "virtual enterprise management has a critical need to automate distributed workflow processes" [Dabke, 1999].

enterprises. The purpose of the architecture is to allow several sub-systems to be connected across the supply chain.

Other authors have referred to this kind of agent-based architecture as a Supply Chain Network (SCN) [Chandra et al., 2000]. A SCN is a society of autonomous agents that address logistics problems. According to its authors, the SCN model is inspired by GERAM and it is composed of two types of agents: the Supply Chain Advisor (SCA) and the Global Supply Chain Advisor (GSCA). There is one SCA for each supply chain member, and its purpose is to provide information about the allocation and usage of local resources. The GSCA, on the other hand, implements group-level coordination between supply chain members. For example, a SCA may inform the GSCA about the delivery date for certain goods or, conversely, the GSCA may warn a SCA that its corresponding enterprise is not complying with the production schedule.

The intrinsic autonomy of intelligent agents means that agents will ultimately choose to act according to its own reasoning, even if this does not please other agents. This situation has led the SMART project [Jain et al., 1999] to introduce the concept of *spheres of commitment* (SoCom). A sphere of commitment represents the promises and obligations that agents have towards each other. In practice, a sphere of commitment is enforced by a representative agent, which behaves as a group leader; this agent is called the SoCom Manager.

According to the SMART project, the SoCom manager contains pre-defined, abstract spheres of commitment that deal with simple buying and selling, but with different sets of commitments. Agents that represent enterprises can browse these options in order to join a particular virtual enterprise, while the SoCom Manager registers the willingness of those enterprises to play specific roles. As soon as all roles have been assigned, the SoCom Manager creates an instance of the sphere of commitment and, from this point, enterprise agents will communicate directly.

Whereas the SoCom Manager is a centralized mechanism, there have been some efforts towards the design of agent-based systems that support the creation of virtual enterprises in a more decentralized fashion [Rocha and Oliveira, 1999]. This approach requires an electronic marketplace where autonomous agents, representing independent enterprises, can meet each other and cooperate in order to achieve a common business goal; these agents are called Enterprise Agents. A special kind of agent, called the Market Agent, represents the customer needs and it is inserted in the marketplace with the purpose of finding the best enterprise to fulfill those needs.

The Market Agent divides the description of the customer needs into a set of sub-goals. Each sub-goal is expressed in a sub-goal description (SGD), which defines a deadline for achieving that sub-goal, a maximum price that the customer is willing to pay, and a list of domain-specific attributes. Each attribute is assigned a preferable value from the list of possible domain values. In addition, each attribute is assigned a utility value, which determines the importance of that attribute when compared to other attributes. The SGD structure is depicted in figure 5.38(a).

In order to find suitable partners for the virtual enterprise, the Market Agent (MA) formulates a different request for each sub-goal, and submits these requests to all enterprises represented in the electronic marketplace. As shown in figure 5.38(b), the request structure is identical to the SGD except for the utility values, which the Market Agent keeps for its own evaluations. In response to each request message, one or more Enterprise Agents (EAs) will formulate a bid and submit a proposal to the Market Agent. The proposal has the structure shown in figure 5.38(c). The Market Agent will then collect all proposals and select the best bid for each sub-goal, according to the following order of preference [Rocha and Oliveira, 1999]:

1. select the bid with lowest evaluation value, where the evaluation value is given by the average difference between preferable value and the proposed value, weighted by the utility value for each attribute;
2. select the bid with lowest cost;
3. select the bid with shorter proposed time;
4. select the bid which leads to a smaller number of virtual enterprise members.

However, this scenario gets more complicated if a sub-goal attribute imposes a constraint on another sub-goal's attributes. For example, the size of a part may place a constraint on the kind of material it can be built from. In this case, it is possible for an Enterprise Agent to send a constraint message to the Market Agent, specifying the attribute that places restrictions on the values of another sub-goal's attributes. As shown in figure 5.38(d), the constraint message identifies the constraining attribute as well as the possible values for the constrained attributes. If an admissible set of bids cannot be found because some attribute values are mutually incompatible, then a conflict arises.

In order to solve this conflict, the Market Agent sends a message to each of the conflicting Enterprise Agents, using the message structure shown in figure 5.38(e). This informs each Enterprise

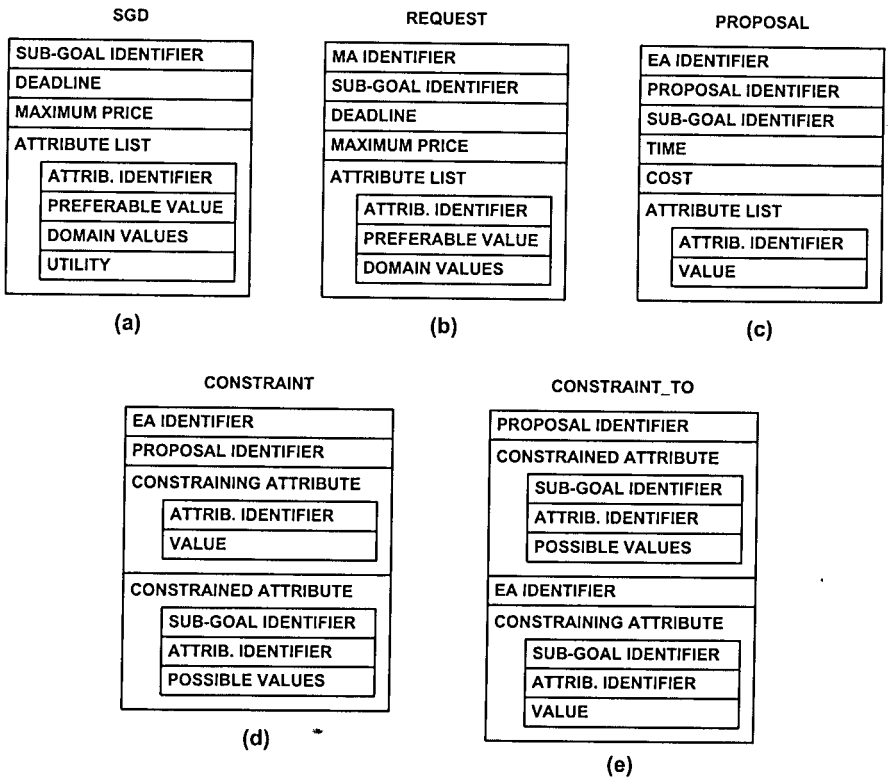


Figure 5.38: Message structures for an agent-based electronic marketplace[84]

[84][Rocha and Oliveira, 1999]

Agent that one of its sub-goal attributes is constrained by another Enterprise Agent's sub-goal attribute (the constraining attribute). The possible values for the constrained attribute are provided, as well as the identification of the constraining Enterprise Agent. From this point onwards, the two Enterprise Agents will engage in a constraint resolution procedure until they agree on a set of admissible attribute values. This constraint resolution procedure is a kind of DCSP algorithm, as discussed earlier in section 5.3.5.2.

5.3.6 Agents and workflow management

In the last few years, there has been a growing trend towards the design agent-based systems that include workflow functionality in one way or another. On one hand, the properties of intelligent agents, such as autonomy and pro-activeness, bring in the need to coordinate the actions of multiple agents, especially if a common goal is to be achieved. On the other hand, the architecture of agent-based systems, which includes features such as intelligent and collaborative behavior, could prove to be instrumental in the design of new capabilities for workflow management systems. Thus, there is a reciprocal relationship between agent-based systems and workflow management systems: agents require coordination capabilities which workflow management can provide, and workflow management systems can be enhanced with intelligent and collaborative behavior provided by software agents.

5.3.6.1 Workflow management in agent-based systems

When compared to other coordination techniques such as rules and conversation protocols, workflow management seems to be the most powerful technique for agent coordination [Griss and Pour, 2001]. Effectively, workflow management systems allow the control, according to an explicitly defined process, of a group of agents that share a common goal. In an agent-based system, however, the workflow participants are software agents, and the process definition binds together the actions of multiple agents. Furthermore, this process definition must be partitioned into a set of *scripts*, so that each script is given to a different agent. The script defines the actions that the corresponding agent must perform. An example of such an approach is the Worklets architecture [Kaiser et al., 1999]: a Worklet is a scripted mobile agent, which travels from host to host, as necessary, until it completes the execution of a given script.

Another example can be found in the Mobile Thread Programming Model (MTPM) [Zhang and Li, 1999], which is an application-level model that simulates the persistence of threads during agent migration from one host to another. According to the MTPM model, each agent, declared as a Java class, must implement three essential methods: dispatch() is invoked just before an agent moves out of the current host, arrival() is called as soon as the agent arrives at the destination host, and run() launches the agent's execution thread. The MTPM model has been demonstrated using Java-specific features, namely object serialization and RMI.

The main issue concerning the MTPM model is that the agent's run() method is implemented by means of a Distributed Task Plan (DTP) [Zhang and Li, 1999], which supports workflow-based control of the agent's execution thread. For this purpose, the MTPM model defines four kinds of *primitives*, i.e., four kinds of possible operations that the agent can invoke within its execution thread. First, mobile primitives allow an agent to migrate to another host, or to clone itself and dispatch one of its duplicates to another host. Second, computational primitives allow an agent to invoke routines that are either carried by the agent or available at the local host. Third, solution synthesis primitives allow an agent to combine its results with those from other agents. And fourth, control primitives

define the sequence of mobile, computational and solution synthesis primitives that an agent must perform. These control primitives include conditional branching and event monitoring primitives, so they can be used to specify a DTP as if it were a workflow process.

In the MTPM model, the functionality required to execute a script, which is equivalent to a process definition, is built into the software agent. In the Worklet architecture, on the other hand, the agent contains only the script, whereas the executing functionality is provided by the underlying agent platform. Another alternative is to implement mechanisms that allow agents to obtain workflow functionality at run-time, and to use that functionality in a plug-and-play fashion [Chen, 1999]. In this case, whenever a process instance needs to be created, an agent will download a workflow service and request this workflow service to create and manage the execution of that process instance. This may involve other agents, and even human users, as shown in figure 5.39.

It is assumed that the plug-and-play workflow service comprises two servers. The Process Manager (PM) controls the flow of tasks, and the Work Manager (WM) takes care of dispatching tasks to other agents (A1 and A2) and to human users (H1 and H2). Whenever agent A needs to create an instance of process P, it will launch the dynamic agents PM and WM, and request them to download their respective workflow servers from some location. In addition, agent A will launch the dynamic agents A1 and A2, which will carry out program tasks during the execution of process P, and it will request these agents to download a message interpreter so they can receive tasks from WM. Once this plug-in phase is complete, agent A will request agent PM to initiate process P. The WM agent dispatches manual tasks to human users via the Web browser, and program tasks to agents A1 and A2, requesting them to download any necessary programs first and then execute them. Task outputs are sent back to the WM agent, and then to the PM agent for flow control.

5.3.6.2 Agents in workflow management systems

Including workflow management functionality in agent-based systems provides a mechanism to synchronize the actions of multiple agents. According to this approach, each agent becomes responsible for a specific task within the logic of a given workflow process. But then, this kind of coordination approach is precisely the point of workflow management, so this suggests that it should be possible to design workflow management systems based on software agents. The idea of designing agent-based workflow management systems is currently being explored from several different perspectives.

One design perspective, to begin with, is to formally describe workflow models and then to derive relevant *actor skills*, which will be provided by software agents [Hannebauer, 1999]. In this case,

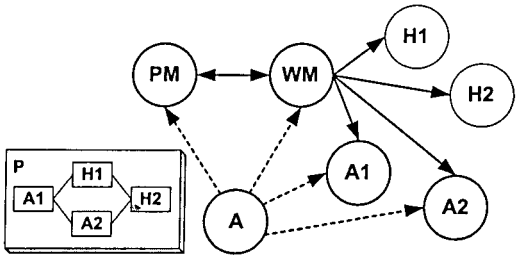


Figure 5.39: Dynamic workflow service provisioning[85]

[85][Chen, 1999]

process models are described using Petri nets, in which transitions represent workflow activities. Each activity is to be performed by a given actor, which must possess certain skills. In order to define these skills, the tasks in which the actor participates, as well as the data and resources the actor makes use of, must be considered. This will lead to one or more UML Use Case Diagrams, which describe the relationships between actors and their skills. The required system functionality can then be specified by marking those actor skills that will be automated by software agents.

At the implementation level, object-oriented techniques are again useful. One possible approach is to encapsulate the system functionality into components called Work-Agents and Work-Servers [Budimac et al., 1999]. A Work-Agent is a mobile agent with the required skills to perform a specific task. All Work-Agents have a common set of attributes (such as the work identification and its deadline) and methods, including a method to save the work to a system file before migrating to another host. At each host there is a Work-Server, which waits for incoming agents and activates them whenever they arrive. The Work-Server also saves the execution state of all agents before the local host is shut down, and loads that state once the system is brought up again. A third component, besides Work-Agents and Work-Servers, is the Work-Host. The Work-Host gives the user basic information about all Work-Agents that currently reside at the local host, and allows the user to select and interact with one of those agents at a time.

Another approach, called agent-oriented system approach [Inamoto, 1999], relies on an agent-based architecture comprising *e-form agents*, *worker agents*, and a single *manager agent*. E-form agents play a similar role to a paper form in paper-based offices, but they have mobility and autonomy. Like a paper form, so too does the e-form agent move around through several work places, but with several enhancements: for example, the e-form agent is able to travel by its own, to react to external events, and to maintain its internal state. Work agents, on the other hand, are stationary agents at work sites, where they can fill the roles of workers or act as proxies to human users. The manager agent is a central, stationary agent at a manager site: it creates e-form agents, transfers knowledge to them, and dispatches them to worker sites. Each e-form agent travels over the network from one worker site to another and returns to the manager site for reporting. This system has been implemented using the MonJa agent platform.

Some authors have realized that the design of new workflow management systems based on software agents is likely to incur high reengineering costs, so they have proposed an *agent-enhanced approach* for workflow management systems, rather than an *agent-based approach* [Judge et al., 1998]. According to the agent-enhanced workflow approach, an existing workflow management system can remain in its place and, on top of this workflow layer, an agent layer is created. This agent layer includes a single Central Administration agent (CA agent) and one Work Processing Center agent (WPC agent) for each work site. The CA agent manages the overall distribution of work by assigning each workflow activity to one of the work sites, using a strategy based on the *contract net protocol* [Sandholm and Lesser, 1998]. Basically, the CA agent announces a work item as a contract to all work sites, and it expects the contractors to return bids for the contract. Next, the CA agent will award the contract to contractor that has submitted the winning bid.

The contract net protocol is just an example of how activities can be assigned to the available resources; other strategies could be implemented as well in an agent-enhanced workflow approach. In general, a work site, a resource, or a work processing center, can all be viewed as providing services required to accomplish certain tasks. The main feature of agent-enhanced workflow is that the agent layer takes care of task assignment, i.e., it takes care of service provision on behalf of the workflow management system. Hence, even if the assignment strategies change, or if new services become available, the same workflow management system can be used without modifications.

The agent-enhanced workflow approach has been extend to support inter-enterprise workflow management as well [Shepherdson et al., 1999a]. In this scenario, each enterprise has its own workflow management system enhanced by an agent layer, as in the regular agent-enhanced workflow approach, but it is assumed that each of these systems makes use of its own ontology. Hence, there has to be a broker agent for each enterprise in order to translate information from the local ontology into a global ontology that can be understood by other workflow management systems. These broker agents communicate with each other using FIPA ACL and the Process Interchange Format (PIF) [PIF, 1996] as the content language, and the system has been implemented using the ZEUS agent platform. There is also a global process monitoring tool that requests descriptions of any local processes that are involved in the overall inter-enterprise business process. The monitoring tool then assembles the various sub-process definitions into a global process, and it performs consistency checks on this process using the Edinburgh Concurrency Workbench [Moller and Stevens, 1997].

The agent-enhanced workflow approach demonstrates flexibility in service provision, but software agents also bring flexibility regarding the processes that can be enacted. In particular, mobile agents can be employed to support ad-hoc workflow processes [Meng et al., 2000]. In this approach, an ad-hoc workflow management system launches a mobile agent whenever a process instance is created, and the mobile agent travels through different locations, where special services are available. At each location, the mobile agent invokes services locally, instead of requiring those services to communicate remotely with the workflow management system. Once a service is performed successfully, a business rule is evaluated in order to determine the next service to be executed. This business rule is equivalent to a transition from one activity to another; the novelty is that this transition is only known at run-time, rather than being pre-defined.

In an enterprise environment there may be fixed locations for each service. But if the desired services are to be found somewhere on the Web, then the mobile agent must contact a broker in order to find the provider of the next service and its location. The broker may choose one service provider from several ones, if multiple service providers are available. This assumes that service providers must be willing to register themselves with the broker agent. If one service fails to execute successfully, the mobile agent may request the broker agent to choose another service provider. In this case, there is also a run-time transition, but this time to a second attempt of the same workflow activity.

Internally, the mobile agent makes use of a set of ECA rules that determine what the agent will do next. For each workflow instance, and besides the mobile agent, there is also a static agent that remains attached to the workflow management system, and whose purpose is to monitor the actions of the mobile agent. The mobile agent will communicate with the static agent periodically during service execution, and it will return to the workflow management system when there are no more services to execute in the current workflow instance. This ad-hoc workflow management system has been implemented using the Voyager agent platform, and a simplified rule engine was implemented with Java [Meng et al., 2000].

Besides allowing processes to be defined at run-time in a completely ad-hoc manner, software agents may also be employed to support the evolution of distributed workflow models [Wang, 2000]. In a hierarchical organization, where processes are distributed into smaller parts called *process fragments*, each process fragment may have to be modified locally at each worker's site due to uncertainties or changes in the way they perform their sub-processes. In a simple organization hierarchy having just three levels, the General Manager gives each Project Manager a fragment of its own process, and the Project Managers delegate to Project Members a fragment of the process they received. Each of

these participants may have to change its own process fragment, which then becomes inconsistent with those of other participants.

Software agents can assist in ensuring that process fragments remain consistent, both vertically across the hierarchy levels and horizontally within each level. For example, if both Project Managers are free to change their process fragments simultaneously, then two *negotiation agents* must be activated in order to safeguard the dependencies between both process fragments. However, if Process Managers can only insert changes one at a time, then an *update agent* is enough to communicate changes back and forth between Process Managers. Another alternative is to have a *voting agent* which communicates changes to other participants and collects their votes in order to determine whether the changes should be adopted or not.

The ability of agents to react to run-time conditions suggests that they could be used to cope with run-time exceptions during workflow execution [Huhns and Singh, 1998]. The number of possible exceptions and contexts in which exceptions may arise is so large that it is impossible in practice to define all exceptions in advance. In an agent-based system, on the other hand, agents can hypothetically re-assign tasks to other resources, establish contracts with optional services, or take alternative courses of action in case of failure. Provided with learning abilities, agents could also become aware of repetitive run-time conditions, such as the fact that a certain resource usually fails within a time period, or the compensation activities that must be usually performed in response to a given exception.

The mobility of software agents can also be shown to improve the performance and scalability of workflow management systems [Yoo et al., 2001]. For this purpose, two basic approaches must be distinguished. One approach is the traditional client-server approach, in which the workflow engine remotely requests each task performer to execute a given task, until all tasks have been completed according to the process definition. The second approach is to launch one or more mobile agents containing the process definition; these agents will visit each work place and request locally the execution of those same tasks. Using stochastic Petri net models, it can be shown that the second approach requires less computational and communication overhead as the number of workflow activities increases within a process definition, or as the number of simultaneously running process instances increases [Yoo et al., 2001].

In conclusion, this discussion suggests that agent-based workflow management systems, should they prove to be feasible, would have several advantages in comparison with their traditional client-server counterparts, namely:

- resemblance of system design to organization [Hannebauer, 1999];
- coupled task state and execution [Inamoto, 1999];
- flexibility in service provisioning [Judge et al., 1998];
- process adaptation [Meng et al., 2000] and evolution [Wang, 2000];
- exception handling [Huhns and Singh, 1998];
- improved performance and scalability [Yoo et al., 2001].

5.3.6.3 Agents, workflow management, and the virtual enterprise

Because agent-based systems require coordination, and workflow management systems strive to improve their flexibility, both of these paradigms are likely to benefit from each other. This idea has

encouraged several authors to devise approaches that include features from both of these paradigms. The agent-workflow decomposition model [Yongjie, 1997] is an illustrative example of this trend. According to this approach, a business process is described as a sequence of workflow activities, which may represent manual and program tasks. The main issue is that each of those workflow activities is managed by an *agent-workflow*, rather than being directly assigned to a resource.

The agent-workflow regards the workflow activity as comprising one or more smaller steps, all of which must be performed in a special order. The agent-workflow interacts with the assigned resource and controls the execution of those steps. Once these steps have been completed, the agent-workflow notifies the main workflow management system so that process execution proceeds to the next activity, i.e., the next agent-workflow is brought into action. This approach employs simultaneously two reciprocal relationships between agents and workflow management: at the process level, workflow management coordinates the actions of agent-workflows, and at the activity level the agent-workflows control the workflow between individual activity steps.

Extending this approach from an enterprise environment to an inter-enterprise scenario, process-level workflow management could manage the interaction between services provided by different enterprises, while activity-level agent-workflows would control the sequence of tasks for each service. Whether in this or in another configuration, workflow management and software agents are indeed becoming the two main ingredients in infrastructures for virtual enterprises. For example, in the MIAMI project [Broos et al., 1999], virtual enterprise members are connected by means of an Active Virtual Pipe (AVP) which contains a Generic Workflow Management Facility and a set of agents to monitor and enforce process execution. Another example is the SBD program described in section 5.3.5.3 on page 228, which has designed an agent-based infrastructure for enterprise integration and virtual enterprises, having a Workflow Agent as one of the main components in its Core Processing System.

The TuCSoN coordination infrastructure [Ricci et al., 2001] aims at coping with two main challenges concerning the management of virtual enterprises: one is the integration of heterogeneous resources belonging to different members, and the other is the execution of business processes that are specific to each virtual enterprise. Thus, a virtual enterprise infrastructure must be able, on one hand, to promote a conceptual homogeneity of resources and, on the other hand, to support the dependencies between business processes and tasks performed at different enterprises. In the TuCSoN coordination infrastructure there is one agent for each task, and these agents are activated according to a set of workflow rules. Basically, these rules specify which agent should be launched when another agent completes its task.

In the Adlet System [Chrysanthis et al., 1999] a single agent, called an *adlet*, is provided with the necessary code, data and credentials for the execution of an entire process instance. An addlet is a mobile workflow agent that visits each service provider according to a trip plan. Adlets can delegate some of their work into other adlets, and they can dynamically change their trip plan to include additional tasks or to drop existing ones. On the other hand, adlets can only travel between nodes where an Adlet Manager is available. The Adlet Manager authenticates adlets and maintains information about the services provided by adlets available on that node.

In a more thorough approach, [Ouzounis, 2001] has developed an agent-based platform for the management of dynamic virtual enterprises, which provides a set of facilities supporting the registration, retrieval and execution of business processes. On a first phase, called Business Process Specification and Registration, an enterprise specifies its business processes by means of an XML-based process definition language, and registers those processes on a central repository. This repository has two different components: the Service Type Repository (STR) stores a list of available service

types, while the Service Offer Repository (SOR) stores the process definitions. The Service Type Agent (STA) manages the Service Type Repository by adding, removing, listing and modifying service types. In the Service Offer Repository, the Service Offer Agent (SOA) manages the storage of process definitions and the Service Offer Retrieval Agent (SORA) is able to retrieve offers based on some constraints.

On a second phase, called Business Process Management, it is assumed that an external customer communicates purchase needs to a VE Representative, which will identify the required business processes and select the appropriate members for a new virtual enterprise that can fulfill those purchase needs. Using negotiation performatives, these members are able to propose, refuse or agree on a set of process parameters before starting process execution. The Requestor Negotiation Agent (RNA) and the Provider Negotiation Agent (PNA) represent both sides in each negotiation. During process execution, the Workflow Provider Agent (WPA) takes care of invoking several Resource Provider Agents (RPAs), while keeping the VE Representative informed about the current process status. All agents are FIPA-compliant, and they have been implemented using the Grasshopper platform.

Besides being employed in the underlying run-time infrastructure, software agents and workflow management have been the basis for a virtual organization development methodology [Zhuge et al., 2002]. According to this methodology the basic elements of a virtual organization are agents, which have: an interface, a reasoning mechanism, a rule set, and a responsibility set. These agents are organized according to a federation hierarchy with three levels: the super-federation level, the federation level, and the agent level. The super-federation level corresponds to the virtual organization top management, while each federation represents a single enterprise, where several agents perform workflow tasks depending on their responsibility set. In fact, each level has its own workflow process, which can be regarded as a sub-process of the next upper level. Hence, a super-federation manages a workflow process across different federations, each federation controls a workflow process across several agents, and each agent may have its own set of workflow steps to perform. This methodology maps existing organizations onto a federation-agent-workflow framework, and then derives the run-time support mechanisms required for the resulting virtual organization.

5.4 Web services

The idea of having intelligent software agents traveling through the network, collecting information and negotiating trade conditions on behalf of a human user is especially appealing in inter-enterprise business scenarios. In a global marketplace such as the World Wide Web, a software agent acting on behalf of an enterprise could search for, evaluate and select potential business partners, while releasing human resources from routine tasks and relying on them just to attain the final decision about which partner should be contracted. However, this idea faces a major impediment. As Tim Berners-Lee and others explain in a thought-provoking article [Berners-Lee et al., 2001], most content available on the Web is appropriate for humans to read, not for computer programs to manipulate in a meaningful way. It is true that Web browsers can parse Web pages in order to determine how to display them but, in general, there is no reliable way for computer programs to ascertain and reason about their semantic content. As a result, software agents are unable to make proper use of the resources available on the Web.

The practical solution to this problem is not to devise exceedingly intelligent programs that can understand the currently available information, but to publish content and provide functionality that will be amenable to being processed by computer programs. In other words, content on the Web

should evolve from multimedia content into information-providing services. These information-providing services are called Web services. Basically, a Web service performs a certain function which can be anything from a simple request to a complete business process. Web services are said to be deployed on the World Wide Web, where other computer applications can discover them and invoke them. Furthermore, Web services are modular applications that can be used as component services to build more complex services.

The concept of Web services has been received with enthusiasm by the software development community, as large software vendors produced their own visions of how Web services should be implemented: Microsoft introduced its own perspective about Web services with its .NET platform, IBM produced software development tools for Web services, and Sun Microsystems proposed the Sun Open Network Environment (Sun ONE) as a complete platform for deploying Web services. Fortunately, these contenders have realized that standards are critical to the success of Web services, and they have adhered to a common set of specifications. In fact, the original purpose of Web services, which is the ability to interoperate with any information-providing service available on the Web, requires global standards. The set of adopted standards, which include SOAP [W3C, 2000b], WSDL [W3C, 2001] and UDDI [OASIS, 2002], is what is known today as the Web Services technology[86].

5.4.1 General architecture

Regardless of any particular service platform, it is common understanding that a Web Service is any information service available on the Web that exchanges XML messages with its clients using Web protocols [Cerami, 2002]. Additionally, a Web Service has two fundamental properties: it must be self-describing, and it must be discoverable. The self-describing property means that a Web Service must be published together with at least some human-readable description of how it can be used. Ideally, the Web Service should include a machine-processible interface description, which identifies all public methods, method arguments, and return values. The discoverable property means that there should be a simple mechanism that allows interested parties to locate the service and retrieve its public interface.

There are three main players in a Web Service architecture: the service provider, the service client, and the service registry. These are depicted in figure 5.40. On a first step, the service provider registers its service on a service registry. The registry stores the name of the service provider, as well as the URL for the service description. On a second step, a service client connects to the service registry and performs inquiries about the available service providers and their services. Assuming that the service client has found the desired service, it will connect to the service provider and retrieve the service description. The service description contains all the necessary details about how to connect to the service implementation, so the client can automatically invoke the service.

In order to implement this behavior, several protocols must be defined. First, there has to be protocol for publishing and finding information about services in the service registry. Second, there has to be a standard way of describing a service in a precise manner so that any client can invoke it. Third, interaction with a service requires messages to be encoded in a specific format, so that the exchanged messages can be understood at both ends. And fourth, in order to exchange messages there has to be a commonly agreed transport protocol. Together, these specifications represent the four layers of the Web Services architecture, as shown in figure 5.41. Figure 5.41 also contains the

[86] In this text, Web Service and Web Services (written with capital "S") refer to a particular Web services technology based on a set of specifications including, but not limited to, SOAP, WSDL, and UDDI. There is another approach to Web services, which is based on the DAML family of Semantic Web markup languages [McIlraith et al., 2001].

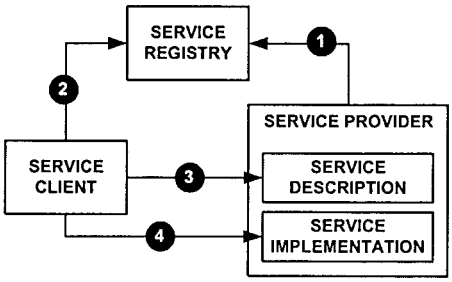


Figure 5.40: Main players in the Web Services architecture[87]

protocols to be used at each layer; these are not the only possibilities, but they are definitely the most common options.

5.4.1.1 SOAP

The Simple Object Access Protocol (SOAP) [W3C, 2000b] is an XML-based protocol that enables client applications to connect to remote services and invoke remote methods over HTTP. The SOAP specification has two major parts. The first part defines the message structure, which encapsulates the service-specific data to be transferred, including the method name, method parameters, or return values. The second part defines how to encode basic data types to be used in those elements by means of a set of conventions, which follow similar rules to the XML Schema specification [W3C, 2000c]. SOAP also defines the precise message structure and fault codes to be used in the event of an error such as, for example, a client trying to invoke a method that the service provider does not implement.

The SOAP message structure is shown in figure 5.42(a), which is somewhat similar to the BizTalk message format shown earlier in figure 5.18 on page 191, since the BizTalk framework makes use of SOAP. Every SOAP message has an envelope, an optional header, and a body. Usually, the envelope contains namespace declarations that are particular to SOAP; for example, one of these namespaces refers to the SOAP version being used. The optional message header provides a flexible way to implement additional functionality, such as authentication or transaction management, without requiring changes at the messaging layer. Message elements within the header portion may include two attributes: the actor attribute specifies the intended recipient for the header entry (which may be one

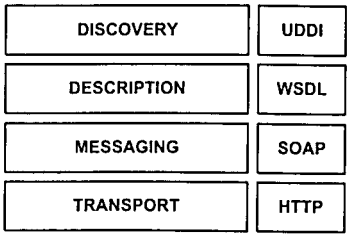


Figure 5.41: Layers of the Web Services architecture[88]

[87] adapted from [Cerami, 2002]
[88] [Cerami, 2002]

of the intermediaries the message goes through), and the mustUnderstand attribute indicates whether the header entry is mandatory or optional for that recipient to handle.

The message body contains the data to be transferred, which comprises a collection of elements that are dependent upon the particular service being invoked. Each body element is identified by a fully qualified name, and it usually includes its corresponding namespace as an attribute value. SOAP specifies only one body element, which is the Fault entry shown in figure 5.42(b), to be used for reporting errors. Within this element, the faultcode element contains a text code that identifies the kind of error, the faultstring element contains a human-readable explanation of the fault, the faultactor element is a URI identifying the party who caused the fault to happen, and the detail element carries application-specific error information.

5.4.1.2 WSDL

The Web Services Description Language (WSDL) [W3C, 2001] is a specific language for describing Web Services in a standard way, so that client applications are able to invoke them. With WSDL, a client can locate a Web Service and understand its public interface, the data types to use in request and response messages, and how to encode these messages with SOAP. Figure 5.43 shows the main elements of a WSDL service description. These elements are encapsulated within a root element called definitions. The elements on the left-hand side concern the messages to be exchanged with the service, as well as the service location. On the right-hand side, the binding element specifies how those messages must be encoded with SOAP.

A WSDL description may contain one or more message elements, whose purpose is to specify the data items to be included in the respective messages. For each of these data items there is a part element, which contains its name and type. There may be several part elements within a message element, i.e., each message may require one or more data items, which correspond to method parameters or return values, depending on whether the message is a request or a response message. This

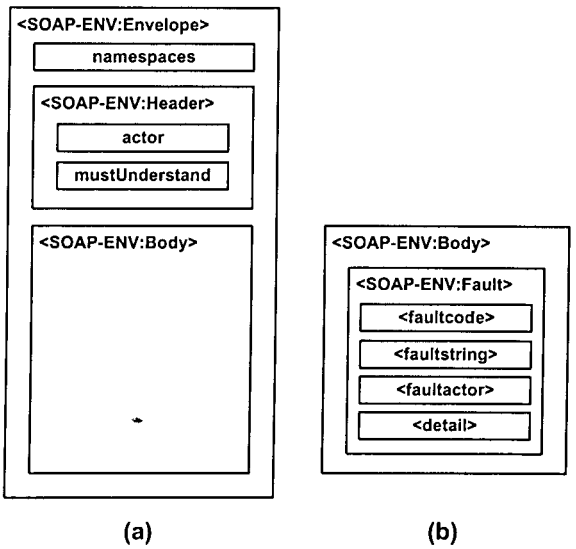


Figure 5.42: General structure for SOAP messages

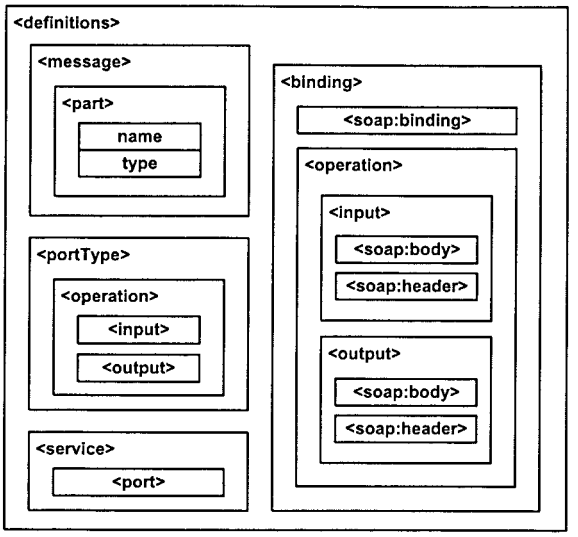


Figure 5.43: General structure for a WSDL service description

is specified by the portType element, which defines how messages are used within operations, and determines which messages are input messages and which ones are output messages.

Given that each input or output element may or may not be included, and that they may be included in any order, the WSDL language is able to describe at least four interaction modes. If only the input element is present, the service receives one message but does not return any response, so this mode is equivalent to a one-way operation. If the service receives one input message and returns one output, then this is referred to as request-response. Alternatively, the service may send a message and expect a response from the client: this mode is called solicit-response. The fourth mode is notification, which is defined as an operation with a single output message, from the service provider to the client.

Going back to figure 5.43, the service element contains one or more port types, which specify the address for invoking the operations defined within a particular portType element. In most cases there will be a single port element, but if the service has more than one set of operations, i.e., if it has several portType elements, then each of these elements may be bound to a different URL.

The last main element in a WSDL description is the binding element, which is used to specify how operations, and their messages, are to be encoded according to a particular messaging protocol. Although the WSDL specification includes bindings for SOAP, HTTP and MIME, SOAP is expected to prevail over other protocols. In this case, the binding element contains a soap:binding element, as depicted in figure 5.43. The soap:body and soap:header elements specify how message parts appear inside a SOAP header and body, respectively.

5.4.1.3 UDDI

The Universal Description, Discovery and Integration (UDDI) [OASIS, 2002] is a technical specification for discovering Web Services. Its purpose is to specify how to publish and find services in a service registry. The UDDI specification addresses the internal data structures for a service registry, as well as the API for accessing it. Major companies such as Microsoft, IBM and SAP have created *operator sites* that implement the UDDI specification. These operators synchronize their data on a

regular basis in order to maintain a global service registry. Thus, a UDDI service registry is a logically centralized directory, though physically the data may be distributed across several databases. It is also possible to set up private UDDI registries, where all the Web Services available within a single enterprise, or within a restricted group of business partners, are registered.

According to UDDI, the information in a service registry is basically a set of instances of four data structures: the businessEntity, the businessService, the bindingTemplate, and the tModel. The relationships between these data structures are shown in figure 5.44. The businessEntity data structure contains institutional information about the service provider, such as its name, address, and contact information. The businessService data structure describes the purpose of a single Web Service, or of a group of related Web Services, in a human-readable way. The bindingTemplate contains information about where the service can be found (e.g. its network address) and how it can be accessed (by HTTP, FTP, e-mail, fax, or even touch-tone phone call). The tModel, or technical model, refers to technical information, such as an external specification, which describes how a client application can interact with the service; one possibility is to have a tModel refer to a WSDL service description.

The instances of these data structures are all stored separately in a service registry, and they are accessed by means of unique identifiers called *keys*. Every instance of a data structure is assigned a unique key at the time it is first published. These keys are used to implement the relationships shown in figure 5.44. In a containment relationship, the containing structure contains an additional entry, which is the key for the contained structure. References are implemented in the same way, but there is an important difference: while there may be multiple references to the same instance, a containment relationship implies that the contained instance is only known to its containing instance. Thus, no two businessEntities may contain the same businessService instance, and no two businessService instances may contain the same bindingTemplate, but several bindingTemplates may refer to the same tModel.

UDDI specifies an API that allows service providers to publish instances of these data structures and allows client applications to search for and retrieve those instances. Hence, this API is divided into the publishing API and the inquiry API. Figure 5.45 lists the registry operations supported by the Microsoft UDDI Business Registry (UBR). All these operations work on a request-response basis. Within the inquiry operations shown in figure 5.45(a) there are operations for searching data structure instances (Find...) and for retrieving their details (Get...Detail). Within the publishing operations, figure 5.45(b) shows operations for inserting (Save...) and removing (Delete...) instances in the business registry. The GetRegisteredInfo operation returns an abbreviated synopsis of all information currently managed by a given service provider.

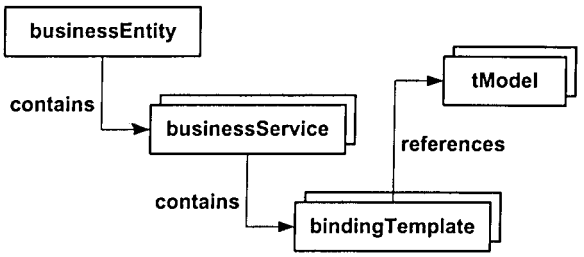


Figure 5.44: Relationships between the main UDDI data structures

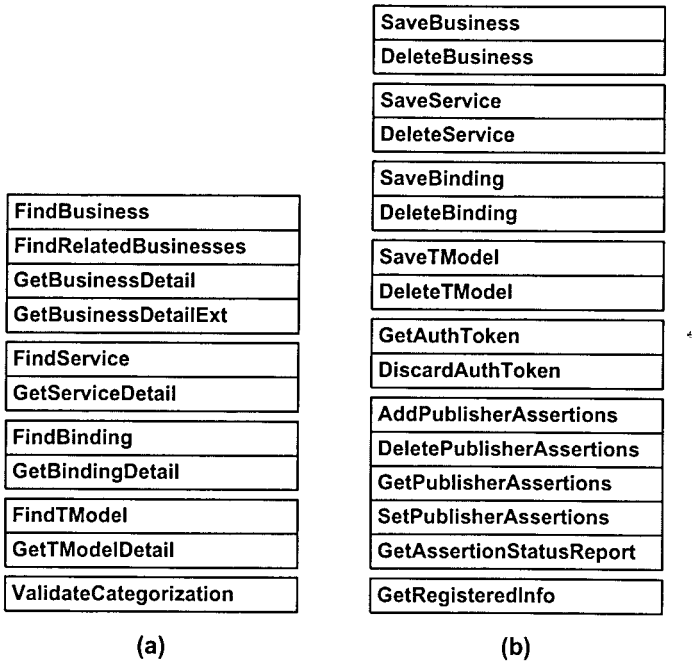


Figure 5.45: Inquiry (a) and publishing (b) operations supported by Microsoft UBR

An interesting feature of UDDI, not shown in figure 5.44, is that it is possible to establish generic relationships between businessEntity instances. For example, a company may wish to express the fact that it is owned by another company, which has also published Web Services in the same registry. To express this fact, one of these parties must create a publisherAssertion, which is another kind of data structure, and it must publish this assertion in the registry. The publishing API shown in figure 5.45(b) contains five methods to deal with publisher assertions. On the inquiry API, the FindRelatedBusinesses operation allows a client application to retrieve all the businessEntity instances that are related to a given businessEntity by means of a publisherAssertion.

Another interesting feature of UDDI is that any businessEntity instance may include optional attributes that identify the service provider within one or more business categories. Currently, the supported categories are those defined in the North American Industry Classification System (NAICS), in the Universal Standard Products and Service Classification (UNSPSC), and in the standard taxonomy ISO 3166 [Cerami, 2002]. Basically, these taxonomies employ a hierarchical classification mechanism, which goes from industry sectors down to individual product and service codes. This way it is possible to browse the contents of a UDDI registry according to each of these taxonomies. The ValidateCategorization operation shown in figure 5.45(a) is used to verify that a specific category exists within a given taxonomy.

Regarding the publishing API, UDDI defines a simple mechanism to control permissions to perform publishing operations. For example, it must be ensured that one service provider is unable to modify or delete information published by another service provider. When logging into the UDDI registry, a service provider will invoke the GetAuthToken operation, while passing its user identifier and whatever credentials it may have as input parameters. The registry will return an authentication token that the service provider must use in order to invoke the remaining publishing operations.

Therefore, the service provider is authenticated every time it calls a publishing operation. Once the desired operations have been performed, the service provider invokes DiscardAuthToken in order to inform the UDDI registry that the authentication token is no longer valid, and the UDDI registry will refuse any operations from anyone who provides the same token.

5.4.2 Additional specifications

The three specifications SOAP, WSDL and UDDI are the foundation for Web Services. Together, they define how to publish and find Web Services, how to describe their interfaces, and how to exchange messages with them. However, standardization in this field is far from complete, as additional specifications come into play [IBM, 2002]. On one hand, it was realized that current specifications still lack important features concerning messaging, security and reliability; these are mandatory improvements within the architectural layers shown in figure 5.41. On the other hand, there is still room for improvements that extend the Web Services architecture with additional capabilities, such as coordination.

Improvements to Web Services messaging include the Direct Internet Message Encapsulation (DIME) [Nielsen et al., 2002b] and the Web Services Attachments (WS-Attachments) specification [Nielsen et al., 2002a]. DIME is a binary message format that can be used to exchange any kind of application-specific data. The WS-Attachments specification describes how SOAP messages can carry attachments, and how a SOAP message and its attachments can be encapsulated within a DIME message. Regarding security, the Web Services Security (WS-Security) protocol [Kaler, 2002a] defines the required enhancements to SOAP in order to implement protection mechanisms. These mechanisms are based on security tokens, which are the focus of another specification, called the WS-Security Profile for XML-based Tokens [Kaler, 2002b]. To improve reliability in message delivery, the Reliable HTTP protocol (HTTPR) [Parr, 2002] has been specified, which is intended to replace HTTP as the transport protocol. To enhance service discovery, the Web Services Inspection Language (WS-Inspection) [Ballinger et al., 2001] defines an XML format for aggregating references to all Web Services available within a given site.

Other specifications address coordination capabilities. One is the Web Service Transaction (WS-Transaction) specification [Cabrera et al., 2002], which distinguishes between two coordination types: Atomic Transactions (ATs) and Business Activities (BAs). Atomic Transactions have short duration and are confined to a limited environment; they are usually the focus of transaction processing systems. Business Activities are long in duration, their actions are applied immediately and made permanent, and they are usually the focus of workflow management systems. The purpose of the WS-Transaction specification is to enable both of these kinds of systems to wrap their proprietary mechanisms and become interoperable with Web Services. The other specification devoted to coordination is the Business Process Execution Language for Web Services (BPEL4WS) [Thatte, 2002], which defines a notation for expressing business process behavior as a sequence of Web Service operations.

5.5 Peer-to-peer networking

Basically, the Web Services technology allows network nodes to expose functionality that will be invoked by remote applications; the technology specifies how the available functionality should be described, and how remote applications can discover and interact with Web Services. For these purposes, Web Services rely on centralized mechanisms, notably UDDI. By contrast, peer-to-peer

networking is a technology that aims at taking advantage of services replicated across many network nodes, and that relies instead on decentralized mechanisms, especially multicast search.

Peer-to-peer networking, usually referred to simply as P2P, can hardly be discussed without mentioning file sharing solutions such as Napster or Gnutella, which have reached unprecedented levels of popularity. Whereas Napster has disappeared after a highly-publicized judicial case, Gnutella and others still live on in the pursuit of unrestricted (and potentially illegal) file sharing across the Internet. Meanwhile, the popularity of such solutions - which is expressed by the huge number of users that those solutions have attracted - suggested that P2P technology could be applied in some business scenarios [Laube, 2001], although it has not always been clear just how this could actually be done.

Some authors claim that P2P networking is not entirely new. For example, Intel Corporation claims to have benefited from substantial savings by using, for the past 10 years, a P2P-like system called NetBatch in order to make use of computing resources across geographically-distributed facilities [Clark, 2001]. If one wants to go further back in time, one could even argue that the Internet itself was originally conceived as a peer-to-peer system [Minar and Hedlund, 2001]. This is because the original structure of the ARPANET, which connected only four computers, as described in section 2.1.1 on page 34, effectively resembles the structure of a peer-to-peer network. Regardless of these analogies it is a commonly accepted fact that P2P networking, as it is known today, has been brought into the spotlight by a series of solutions that have unleashed surprising possibilities. Applications of P2P solutions include distributed computing, file sharing, anonymous publishing, and instant messaging. The next paragraphs briefly describe illustrative examples of these applications.

5.5.1 SETI@home

The SETI@home project [Anderson, 2001] is part of the SETI (Search for Extraterrestrial Intelligence) research program, which basically listens to and analyzes signals that emanate from stars and other natural sources in order to detect intelligent life outside the Earth. For this purpose, SETI research relies on radio telescopes that probe the sky for such signals. The largest of these radio telescopes is Arecibo, which is located in a natural hollow in the hills of northern Puerto Rico. In 1992, a secondary antenna was mounted next to Arecibo; this antenna was designed to follow the position of Arecibo but instead of focusing on a particular point, as Arecibo does, it covers a wide area around that point. Therefore, while Arecibo probes a single star, the secondary antenna covers several stars but without loosing too much time with each star. Over long periods of time, the secondary antenna is able to cover the entire band of sky visible from Arecibo.

The secondary antenna produces each day about 50GB of data, which must be transferred to facilities located at Berkeley where, formerly, all frequency analysis took place. Since the network connection between Arecibo's location and Berkeley is not fast enough to transfer such large amounts of data, the data are recorded in digital tapes and then mailed to Berkeley. Berkeley receives around ten tapes from Arecibo each week. Once at Berkeley, the data is divided into small, more manageable work units, each unit having about 300KB. Splitting data from tapes into work units is a computationally-intensive task, so several machines are required in order to accomplish this job, as suggested in figure 5.46. Each work unit is then assigned to a client via the data distribution server.

The main feature of SETI@home is that the frequency analysis of each work unit can be performed by any computer which is permanently or occasionally connected to the Internet. Instead of performing all data analysis at Berkeley, which would require extremely powerful computational resources, SETI@home relies on the willingness of others to make some of their computing power available to SETI. In fact, there are often idle times when a personal computer is turned on with-

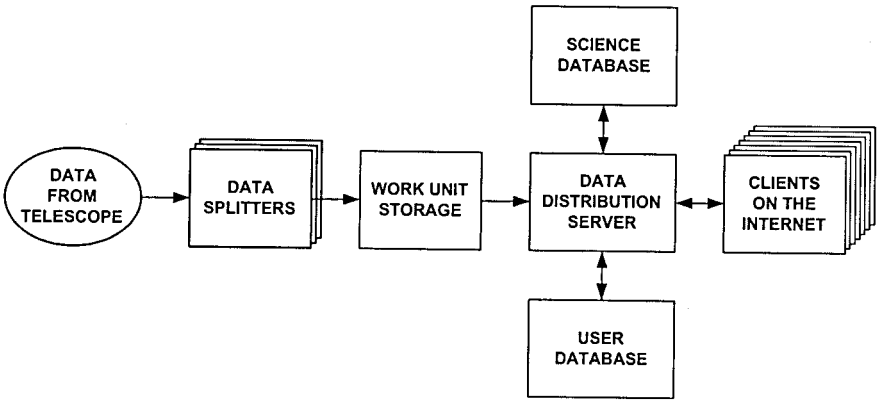


Figure 5.46: High-level architecture for the SETI@home project[89]

out being actually performing any useful job, or at least a job that makes full use of its capabilities. SETI@home was devised to take advantage of this unused computing power.

The SETI@home client application was implemented as a screensaver application, so it runs only when the computer is not being used. When it runs, the client connects to the data distribution server and retrieves a new work unit. The client then performs frequency analysis on the work unit, which may take from hours to days to complete, depending on the computer's capabilities. Every few minutes the client writes the current results to disk so that it can later resume work if the computer is turned off. When the analysis of the work unit is complete, the client connects to the data distribution server and returns back the results. It also fetches a new work unit to work on. The time that clients take to handle a single work unit varies considerably, and some clients may not event return the results for a work unit they have received. Because new data arrives at constant pace and storage space is limited, work units have to be discarded even if their analysis is not complete.

At first, the system comprised just three pieces: the data distribution server, a database server, and a Web server. Over time, the system has been improved with better software routines and an increased number of server machines. What was perhaps most remarkable was the number of users who voluntarily adhered to SETI@home. With millions of clients, SETI@home achieved an aggregate computing power that is faster than the fastest supercomputer currently available [Anderson, 2001]. The relational database that keeps all information about tapes, work units and clients is currently one of the main performance bottlenecks. The authors of SETI@home believe that this kind of peer-to-peer approach unleashes enough computing power not only for SETI but for many other computationally-intensive projects as well.

5.5.2 Gnutella

Gnutella is a file sharing solution based on the Gnutella protocol [Clip2, 2001], which is one of the most paradigmatic examples of how distributed search can be performed in peer-to-peer networks. Knowing some details about the Gnutella protocol is essential in order to understand that a P2P solution, just like any other kind of solution, may require some trade-offs. In Gnutella, performing search in a fully decentralized network comes at the expense of inefficient use of network resources.

[89][Korpela et al., 2001]

On the other hand, the Gnutella protocol specifies some features that are intimately connected with the nature of P2P solutions.

In Gnutella, each node is called neither a client nor a server, but rather a *servent*. A servent can behave simultaneously as a client and server: its client-side interface allow users to issue queries and receive search results, whereas its server-side interface allows the node to accept queries from other servents and return results. It should be noted that these interfaces are low-level interfaces based on TCP/IP connections.

To join the network, a servent must establish a TCP/IP connection with at least one other servent. When the connection is established, the first servent sends a Gnutella-specific connection request string specifying the protocol version to be used. If the second servent accepts the connection, it must reply with "OK". However, the second servent may refuse the connection either because the limit of incoming connections has already been reached, or because it does not support the specified protocol version. In this case, the first servent must try to connect to another servent. The original Gnutella protocol does not specify how a servent may obtain the addresses of other servents; in practice, this is often done by means of host cache services.

The Gnutella search protocol comprises five message types which are called *descriptors*, and it specifies the byte structure for each of these messages. The five possible descriptors are Ping, Pong, Query, QueryHit and Push, and they are depicted in figure 5.47. Descriptors have a header, whose structure is identical for all descriptors, and a payload, whose structure is specific to each descriptor. The descriptor header contains a unique message identifier, a payload descriptor (a numerical value) which identifies the message type, a time-to-live value, a hops count, and the length, in bytes, of the message payload. The time-to-live value specifies the number of times a message can be forwarded to another servent before being discarded, whereas the hops count records the number of times the message has been effectively forwarded. Every time a message arrives at a new servent, its time-to-live value is decreased and its hops count is increased. When the time-to-live value reaches zero, the message will no longer be forwarded.

The Ping descriptor is the most simple one: it has a header but no associated payload. The Pong descriptor contains the port number and IP address to connect to a given servent, together with the number of files and the total number of kilobytes available from that servent. A Pong descriptor is always sent in reply to a Ping, and several Pong messages may follow a single Ping descriptor. When a servent receives a Ping descriptor, it will usually produce a single Pong descriptor containing information about itself; but it may as well produce additional Pong responses containing information about other servents that it knows of. The purpose of Ping and Pong descriptors is to allow a servent to discover other servents in the network.

The Query and QueryHit descriptors are used in a similar request-reply fashion, although for a different purpose, which is to search for files matching a given search string. Since the ultimate goal of a user is to download one or more of these files from another servent, the user may require the source servent to provide a minimum transfer rate. For this reason, the Query descriptor has a "minimum speed" field, which expresses that transfer rate in KB/second. The "search criteria" field contains the search string which, in practice, is usually matched against the file names available at the destination servent.

The QueryHit descriptor is the response to a Query message. The QueryHit descriptor contains the number of files that match the search string, the port number and the IP address of the responding host, the result set, and a servent identifier that uniquely identifies the responding servent on the network. The result set contains the set of matches for the previously submitted Query. The number of matches in the result set equals the "number of hits" value, and each match in the result set is

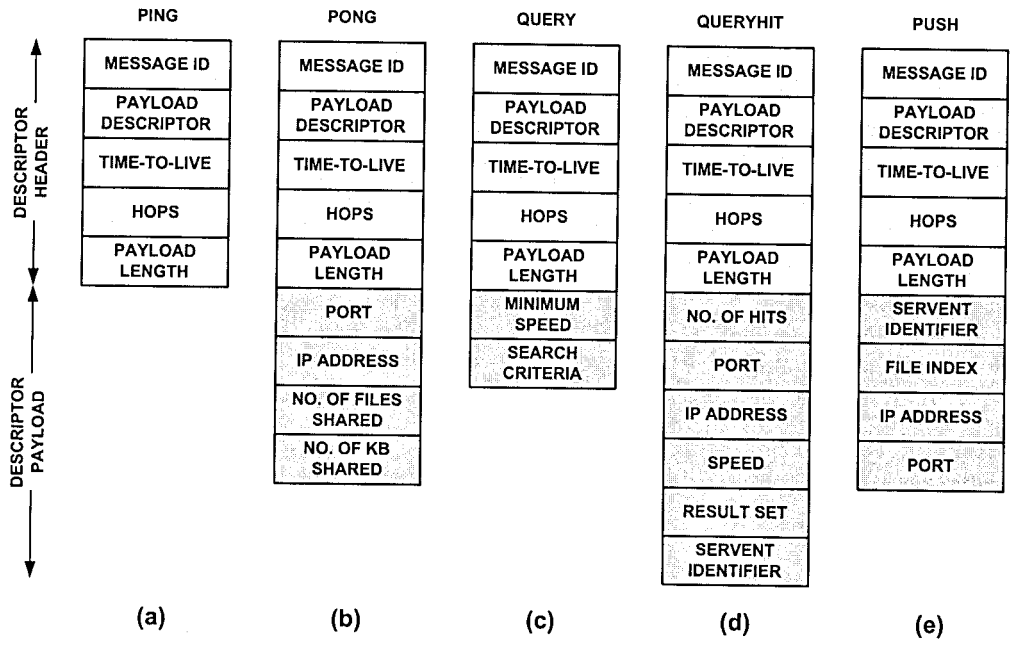


Figure 5.47: The five descriptors for the Gnutella protocol

described by three elements: the file index, which is used to uniquely identify the file at the responding servent, the file size in bytes, and the file name.

After having collected the results from one or more QueryHit messages, the servent may wish to download one or more files that matched the search string. For each file that the servent intends to download, it sends a Push message to the source servent in order to initiate file transfer. The first field in the payload of a Push descriptor ("servent identifier") identifies the source servent, and it must equal the servent identifier from the original QueryHit message. The second field ("file index") is taken from the result set and it identifies the desired file. In response to a Push message, the source servent connects to the IP address and port number specified in the Push message, and initiates the transfer of the specified file.

The Gnutella protocol specifies a set of rules concerning message routing, which are illustrated in figure 5.48:

- *Ping and Pong descriptors* - Whenever a servent receives a Ping descriptor, the first thing it does is to reply with one or more Pong messages, whose message identifier equals the original Ping identifier. Then, the servent decreases the Ping time-to-live value and, if this value is still greater than zero, the servent forwards the Ping to all of its immediate neighbors except to the original sender of the Ping. The neighbors will produce their own Pong responses, and forward the Ping once again if and only if the time-to-live value is still greater than zero after being decreased. When a servent receives the Pong responses from its neighbors it forwards them back to the original Ping sender, but only if the Pong identifier matches the original Ping identifier.

- *Query and QueryHit descriptors* - Whenever a servent receives a Query descriptor, the first thing it does is to match the search string against its local files. If any match is found, the

servant replies with a QueryHit message whose identifier equals that of the original Query. Afterwards, the servent decreases the Query time-to-live value and, if this value is still greater than zero, the servent forwards the Query to all of its immediate neighbors except to the original sender of the Query. The neighbors will produce their own QueryHit responses, if any, and they will forward the Query once again if and only if the time-to-live value is still greater than zero after being decreased. When a servent receives the QueryHit responses from its neighbors it forwards them back to the original Query sender, but only if the QueryHit identifier matches the original Query identifier.

- *Push descriptor* - Whenever a servent receives a Push descriptor, the first thing it does is to check whether the Push identifier is the same as that in a previously sent QueryHit. If not, the servent simply discards the Push message. The reason for this behavior is that an outgoing Push message must be sent along the same path as the incoming QueryHit message. If the Push identifier does match a previous QueryHit identifier, the servent forwards the Push to its immediate neighbors. Those who have not seen before a QueryHit with the same identifier will discard the Push; but the one who has seen it will forward the Push again to its neighbors, and so on, until the Push eventually reaches the source servent that has the specified "servent identifier". When this happens, the source servent opens a connection to the IP address and port number specified in the Push, and initiates the transfer of the requested file.

Alternatively, instead of sending a Push message, once a servent receives a QueryHit message it may initiate a direct HTTP connection to the source servent in order to download one of the files described in the result set. For this purpose, the servent sends an HTTP GET request specifying the file index, the file size, and the range of bytes to download. The range of bytes parameter is intended to allow the servent to resume an interrupted download. When the source servent receives an HTTP GET request it must respond with an HTTP OK header followed by the file data.

From this description, it can be seen that Gnutella relies extensively on the ability of each servent to forward messages to its neighbors. The individual behavior of each peer makes Ping and Query messages propagate throughout the network, up to a maximum number of reachable peers limit imposed by the time-to-live value (T) and by the number of immediate neighbors (N). Ideally, the Query time-to-live value should be infinite, so that the Query is forwarded to all servents, maximizing the likelihood of finding the desired file. However, even for small values of T and N it can be shown that the number of reachable users simply explodes [Ritter, 2001]: for example, for N=4 and T=6

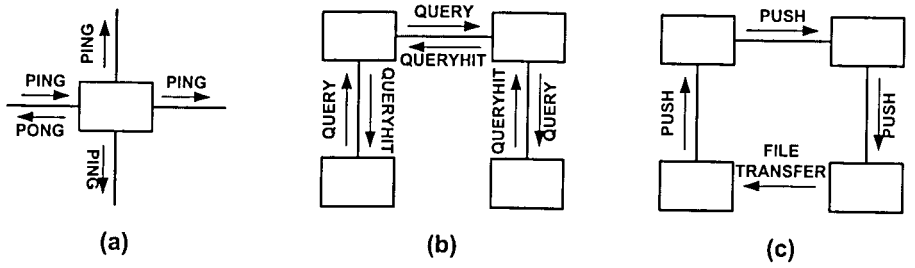


Figure 5.48: Descriptor routing in Gnutella[90]

[90][Clip2, 2001]

the number of reachable peers is already over one thousand, and for N=7 and T=8 it is over two million. This means that issuing a single Query message may result in millions of messages traveling across the network, giving a total amount of data that is many times larger than the file that the user will eventually download. For a large Gnutella network having 50 000 servents it was estimated that the total network traffic, excluding file transfers, is about 1Gbit/second [Ripeanu et al., 2001]. Of course, attempting to reduce bandwidth use by decreasing the number of reachable servents would have a negative impact on Gnutella's search capabilities, since potentially less search results would be obtained.

5.5.3 Freenet

The purpose of Freenet [Clarke et al., 2001] is to allow the publication, replication and retrieval of data while ensuring that both authors and readers can remain anonymous. Freenet is a network of identical nodes which make some of their disk space available in order to store data files. In addition, Freenet nodes are able to route requests to other nodes if they do not hold the file being requested. However, it should be noted that Freenet does not employ broadcast search as in Gnutella, and it does not rely on any centralized index either (as Napster did). The main focus is on ensuring anonymity, so Freenet has been designed in order to make it extremely difficult to discover the origin or destination of a file traveling through the network. In addition, since each node voluntarily provides its own disk space to store files that travel through the network, Freenet has devised encryption mechanisms that make it difficult to held any given node responsible for the data that it stores locally.

In Freenet, each file is identified by a *key*, which a user must obtain or compute in order to request the corresponding file. For the user that publishes the file, there are three possible ways to generate the file key:

- *keyword-signed key* (KSK) - this type of key can be derived from a short description of the file, called the *descriptive string*[91]. First, the user creates a descriptive string for the file, and then it uses a deterministic algorithm to generate a public/private key pair from that descriptive string. By applying a hash function to the public key the user obtains the file key. The private key, on the other hand, is used to sign the file. Additionally, the file is encrypted using the descriptive string as encryption key. This procedure is illustrated in figure 5.49(a). For another user to retrieve the file, all it is required to know is the descriptive string. One limitation of keyword-signed keys is that they form a flat namespace.

- *signed-subspace key* (SSK) - this key allows the publisher to own a namespace[92] so that only this user may publish files under that namespace, possibly according to a hierarchical structure. First, the user randomly generates a public/private key pair which identifies the namespace. Then, the public key and the file's descriptive string are hashed independently, combined via an exclusive-or (XOR) operation, and then hashed together again, which yields the file key. The private key is used to sign the file, which is subsequently encrypted using the descriptive string as encryption key. This procedure is illustrated in figure 5.49(b). For another user to retrieve the file, it must know the descriptive string and the namespace's public key.

- *content-hash key* (CHK) - is useful to implement file updating and splitting. In this case, the file key is derived by simply hashing the contents of the original file. In addition, the file is

[91] An example of a descriptive string is text/philosophy/plato/republic (adapted from [Clarke et al., 2001])
[92] An example of a namespace is text/philosophy/ [Clarke et al., 2001]

encrypted using a randomly-generated encryption key. For another user to retrieve the file, it must know both the file key and the decryption key.

In Freenet, each node has a dynamic routing table that contains the addresses of other nodes together with the keys for the files they are thought to hold. File requests travel from node to node according to the routing table at each node. Each request either results in the file being found or a failure message being generated; in both cases, the result travels the same way backwards. Freenet has devised routing algorithms in order to adjust routes over time in order to improve efficiency. The routing table for each node contains its immediate neighbors (the peers it can connect directly to), and this routing table is being constantly updated as file requests and replies travel through the node. Each request has a *hops-to-live* limit, which is decremented at each node the request travels through; when the hops-to-live reaches zero, the node generates a failure message. Each request has also a unique identifier in order to prevent loops.

To retrieve a file from the network the user must first obtain or calculate its file key, and then submit the request to its local node, specifying a certain hops-to-live limit. When a node receives the request it searches its local data store and, if the file is found, the node returns the file together with a node saying the it was the source for that file. If the file is not found in the local data store, the node looks up in the routing table the most similar key to the one being requested, and delegates the request to the corresponding node. If this second node is unable to find the file, the first node will delegate the request again to other nodes. When a file is found and it is being returned, every node along the way does two things: (1) it caches the file locally, and (2) it inserts a new entry in its routing table. This new entry associates the data source with the originally requested key. However, each node may change the reply message in order to claim itself or any other node as the data source, so that it is difficult for anyone to find out who the data source really was.

Figure 5.50 illustrates an example of a request sequence. When the user submits a file request to node B (step 1), node B walks through its routing table and determines that a file with a similar key is held by node C, so it delegates the request to that node (step 2). However, node C is unable to connect with further nodes, so it sends back a failure message to node B (step 3). Node B then tries a second node which holds a file with the second most similar key to the original key being requested (step 4). Node E delegates the request to node F (step 5) which, in turn, delegates it to node B (step

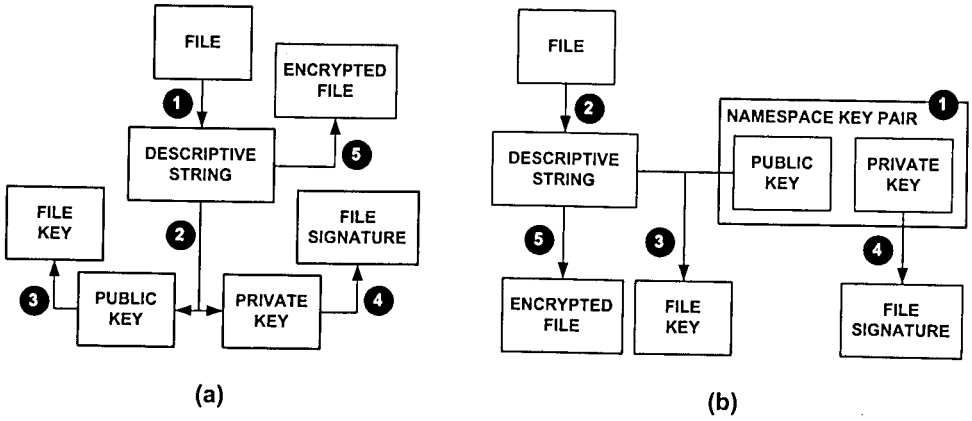


Figure 5.49: Keyword-signed keys (a) and signed-subspace keys (b) in Freenet

6). But node B notices the loop, so it sends a failure message back to node F (step 7), which forwards it to node E (step 8). Node E then tries another node (step 9) which has the requested file. The file is returned from node D to node E (step 10), which forwards it back to node B (step 11), from where it goes to node A (step 12). Throughout this return path, nodes E, B and A cache the file locally and create a new entry in their routing tables for the key request. In each of these nodes, this entry may point to node D, the original data source, or to any other node chosen randomly. Since the data has been replicated in several nodes, subsequent requests for the same file key can be satisfied from any one of these nodes without having to go again all the way to node D.

When a user wants to publish a file, it generates its key and submits the file key to the local node together with a hops-to-live limit. If the key collides with a pre-existing file key, the local node returns the pre-existing file as if the user had actually submitted a request for that file. When the user receives the pre-existing file, it knows that it should choose another key for the new file. If the file key is unique in the local data store, the node looks up the nearest key in its routing table and forwards the new file key to the corresponding node. If the key collides with a pre-existing key at that node, the node returns the pre-existing file, again as if a request had been made for that file. If it does not collide, the new key is forwarded to another node, and so on, until hops-to-live reaches zero. If this happens without the key colliding with a pre-existing one, an "all clear" result is propagated back to the user, who will now send the new file to be cached in all nodes along that route.

The authors of Freenet claim that the publishing and retrieval behavior actually improve the network's overall efficiency since nodes tend to accumulate files with similar keys. In addition, the fact that routing tables are constantly being updated means that the network itself will adapt to requests, increasing the connectivity between the nodes that request files and the nodes that provide them. On the other hand, data replication increases redundancy so only those files that are seldom requested run the risk of becoming unavailable due to node failure or disconnection. However, any file is prone to disappearing, since each node has a limited storage capacity. When this limit is reached, one or

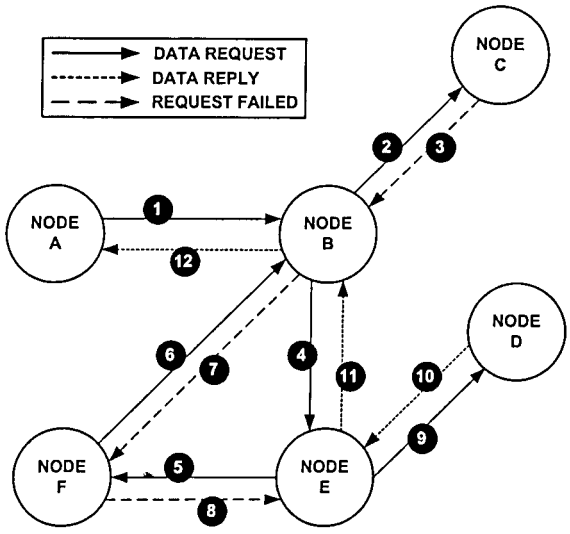


Figure 5.50: An example of a request sequence in Freenet[93]

[93][Clarke et al., 2001]

more existing files may have to be discarded in order to accommodate a new file. This means that the more popular a file is, more likely it is to survive. It also means that authors may have to periodically publish their files in the network.

The fact that files are cached at several nodes throughout the network does not mean that these nodes can be held responsible for the files they store. In fact, every node can deny knowledge of the contents of its data store since the files are encrypted and, in general, a node only possesses the file key, not the encryption key. A user that requests a file knows the descriptive string, therefore it knows the encryption key and it can compute the file key by applying a hash function to the descriptive string. But a node that stores a file knows only the file key, and it would have to revert the hash operation in order to find out the descriptive string, which is the encryption key. The legal protection of nodes is then based on the assumption that, if a node cannot decrypt the file, it cannot know or be held responsible for its contents.

5.5.4 Jabber

The Jabber project[94] has developed on an open, XML-based protocol to support message exchange between any two points on the Internet. The Jabber protocol comes together with a proposed architecture that is not a pure peer-to-peer solution, since Jabber relies on many client/server features. In fact, the Jabber architecture closely resembles that of e-mail. Jabber requires peers to register on and connect to a server, to which they send messages addressed to other peers, and from which they receive messages sent by other peers. To send a message to a peer registered in a remote server, the sender must push the message to its local server, which will deliver the message to the remote server; on its turn, the remote server will deliver the message to the destination peer. In this last step, Jabber differs noticeably from e-mail because it delivers the message immediately to the destination peer if the peer is online. For this reason, it is usually said that Jabber enables *real-time* message exchange.

The primary application of Jabber is instant messaging - the ability of a user to exchange messages with other users as fast as the underlying network connection allows. In this scenario, Jabber can provide very similar functionality to other instant messaging programs such as ICQ or AOL Instant Messenger[95]. However, it should be noted that Jabber was devised to support message exchange not only between human users but also between software applications, or even between human users and "intelligent" services available on the Internet. To support these possibilities, Jabber insists that the messaging protocol should be developed as an open, XML-based protocol in order to promote the exchange of structured content between users. In addition, Jabber allows multiple servers with different users to connect to each other rather than relying on a single central server, as most instant messaging solutions do. The way servers connect with each other - to enable message exchange between their users - resembles a P2P network, where there is no central authority.

When a user or client connects to its server, it opens a one-way XML *stream* to the server, which initiates a new session. In response, the server opens a one-way XML stream to the client, returning a session key. Both streams remain open during the entire duration of the connection. All communication between client and server takes place by sending small snippets or *chunks* of XML within these streams. Basically, an XML stream is an XML document: opening an XML stream is equivalent to sending the opening tag for the document root element; closing the XML stream is equivalent to sending the corresponding closing tag. Within a stream there may be several chunks, which are basi-

[94] http://www.jabber.org/
[95] http://www.aim.com/

cally self-contained XML elements, equivalent to child elements within the document root element, as suggested in figure 5.51. Jabber defines three possible types of chunks:

- *message* - A message chunk represents a single message from one user to another, and its body conveys some structured information whose format may be defined by an external XML namespace. A message chunk may contain also a thread attribute, which is an arbitrary string generated by the sender and that may be included in replies, so that both client and server can keep track of a conversation. (It is interesting to note that this attribute plays the same role as the token color parameter, as discussed in section 7.1.3.3 on page 365.)

- *presence* - The presence chunk is intended mainly to expressed one of several possible status of user availability, such as "away" or "do not disturb"; these are common options in most instant messaging programs. Additionally, the presence chunk allows a client to subscribe to the presence of another user, so that when this user is online the client is notified.

- *iq* - The iq chunk is intended to support simple request/reply interaction: its type attribute specifies whether the iq chunk is a request or a response to a previous request. The content of iq chunks is defined by an external XML namespace.

The Jabber protocol specifies many other features including, but not limited to, user authentication, user registration, subscribing to other users' presence, managing a user's list of contacts (which Jabber refers to as a *roster*), connecting users directly in order to transfer files, exchanging contacts between users, storing messages for later delivery, establishing conversations between multiple users (group chat), and transporting remote method calls as iq chunks within an XML stream.

5.5.5 Project JXTA

P2P solutions such as SETI@home, Gnutella, Freenet, Jabber and others have been developed with specific purposes in mind: SETI@home takes advantage of unused computing power, Gnutella focuses on a distributed search protocol, Freenet puts the emphasis on anonymous publication and retrieval of data, and Jabber aims at enabling real-time message exchange between any two points on the Internet. Each of these solutions illustrates a potential use of P2P networking, and each of

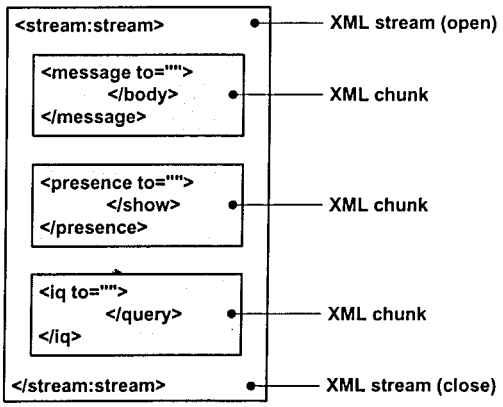


Figure 5.51: XML streams and chunks in Jabber

them has been developed separately and from the ground up. Project JXTA[96] is a different kind of P2P solution: actually, it is a general-purpose P2P platform on top of which P2P solutions can be developed. Basically, project JXTA specifies a set of six protocols [Wilson, 2002] that address the typical behavior of peers in a P2P network. Project JXTA also provides an open-source reference implementation of these protocols.

5.5.5.1 JXTA Concepts

Peers Like Jabber, JXTA also aims at enabling communication between any two points on the Internet. For this purpose, JXTA realizes that there must be appropriate mechanisms to traverse firewalls. But JXTA goes even further: it realizes that each peer must be able to propagate messages to any other peers in the network, regardless of their network location. In fact, JXTA aims at establishing a *virtual network overlay* that allows peers to interact and organize themselves independently of the underlying network topology. For these reasons, JXTA defines three kinds of peers:

- *simple peer* - A simple peer represents an end user, which may be located behind a firewall and, hence, unable to communicate directly with other simple peers outside that firewall.
- *router peer* - A router peer is an intermediary peer whose purpose is to allow simple peers to communicate across a firewall. If necessary, a router peer may use different network transport protocols to connect with each simple peer. On the other hand, each message may have to traverse more than one router peer, if there are several firewalls in the communication path between two simple peers. For a peer to send a message to a second peer, the first peer must determine which router peers it should use. JXTA specifies the protocol that allows peers to find routes to communicate with other peers.
- *rendezvous peer* - The concept of rendezvous peer had to be introduced in order to specify a mechanism that allows peers to broadcast messages to other peers, without having to require the underlying network transport protocol to provide broadcast capabilities. Basically, a rendezvous peer propagates an incoming message to all simple peers it knows of. For this reason, rendezvous peers are also used as meeting places: each simple peer must connect to a rendezvous peer so that it can propagate messages to other peers. This same rule applies to rendezvous peers themselves: each rendezvous peer may connect to another rendezvous peer so that the simple peers connected to them will be able to communicate with each other, thus forming a P2P network. Rendezvous peers are usually placed outside firewalls; if a rendezvous peer is placed inside a firewall it must use either a network transport protocol that is authorized by the firewall or a router peer outside that firewall.

In practice, each peer may simultaneously play the roles of a simple peer and of a rendezvous peer, for example, or of a router peer and of a rendezvous peer, as another example; it may even play the role of a simple peer, a router peer and a rendezvous peer simultaneously. The main purpose of defining these kinds of peers, as well as other JXTA-specific abstractions, is to identify the functions that are required to implement a P2P network on top of any kind of network.

[96] http://www.jxta.org/

Endpoints Each peer is identified by a unique identifier, which is usually referred to as its *peer id*. In addition, each peer has one or more *endpoints*, which encapsulate transport-specific addresses that can be used to communicate with that peer. For example, there could be one endpoint for raw TCP/IP communication, another endpoint for HTTP communication, and yet another endpoint for communication with Transport Layer Security (TLS). Endpoints are defined independently of the underlying transport protocol, so they can represent any kind of network address according to the transport protocols provided by the underlying network infrastructure. To communicate with a peer, one may choose any of its endpoints. However, if one requires some transport features, such as encryption for example, one could prefer the use of one endpoint over another. If the peer has just one endpoint, then there is just a single option to communicate with it.

Peer groups JXTA allows peers to organize themselves into *peer groups*. A peer group is a set of peers with common interests. For example, peers interested in sharing files could belong to one peer group, whereas peers interested in instant messaging could belong to another group. The concept of organizing peers into peer groups is the way JXTA has found to accommodate several possible kinds of services within a single P2P network. In the previous example, the first group could use Gnutella-like services to share files, whereas the second group could make use of Jabber-like services to perform instant messaging. Each peer group may be protected by some membership rules. In its simplest form, membership control is implemented by requiring peers to go through an authentication method before joining the group.

Services A peer group is a subset of the P2P network where special *services* are available. Once a peer has successfully joined the peer group, it will be able to communicate with other peers in that peer group. Within a peer group, there may be two kinds of services available:

- *peer services* - A peer service is some functionality offered by a single peer to other peers; the service is only available when that peer is connected.
- *peer group services* - A peer group service is some functionality provided by several members of the peer group; as long as one member of the peer group is connected, the service is available. In general, a peer group service becomes more useful as more members adhere to the service.

The NetPeerGroup When a peer connects to the P2P network, it automatically joins a global group called the NetPeerGroup, which allows it to immediately communicate with other peers in that group. The peer may remain in this group or, from within this group, it may either create a new peer group or join an existing peer group. If the peer creates a new peer group, it must specify the authentication method that other peers must go through before joining the group; if the peer joins a new group, it must go through the authentication method defined for that group. In general, each peer group has its own *membership service*, which implements the authentication method for that group. JXTA provides a default implementation for such a membership service.

Pipes Once inside a peer group, peers use *pipes* to communicate with the services available within that group (i.e., to communicate with the peers that implement those services). A pipe is a virtual communication channel between two endpoints: both endpoints appear to be directly connected to each other, even if they do not have a direct physical link. A pipe may actually comprise several connections across several rendezvous or router peers. Pipes are asynchronous and unidirectional,

and they support the transfer of any kind of binary data between peers within the same peer group. Each peer has a different perspective about a given pipe: for the sending peer the pipe is an *output pipe*, whereas for the receiving peer it is an *input pipe*. JXTA provides three kinds of pipes:

- *unicast pipes* - A unicast pipe connects exactly two endpoints together: an output pipe on one peer sends messages to an input pipe on another peer.
- *propagate pipes* - A propagate pipe connects one output pipe to several input pipes, so that all input pipes receive the message sent through the output pipe.
- *secure unicast pipes* - A secure unicast pipe is an encrypted communication channel between two endpoints. JXTA provides a default implementation of secure unicast pipes based on TLS.

Advertisements All JXTA elements - peers, peer groups, services and pipes - can be represented by special XML documents called *advertisements*. JXTA protocols make extensive use of advertisements in order to publish and let other peers know of the existence of peers, peer groups, services within a peer group, and pipes to communicate with those services. For each of these elements there is a special kind of advertisement: there are Peer Advertisements, Peer Group Advertisements, Service Advertisements, and Pipe Advertisements. When a peer connects to the P2P network, it publishes its own Peer Advertisement to let other peers know about its presence. When the peer creates a group, it publishes the corresponding Peer Group Advertisement to let other peers know about the group and join the group. When a peer creates a new service within the group, it creates two advertisements: a Service Advertisement that describes the service, and a Pipe Advertisement that describes the pipe to be used when communicating with the service. Then, the peer inserts the Pipe Advertisement into the Service Advertisement and publishes the resulting Service Advertisement. This way, when another peer finds the Service Advertisement, it may extract the Pipe Advertisement and create an output pipe to communicate with the service.

The JXTA layered architecture JXTA combines all the previous concepts in a layered architecture, as shown in figure 5.52. On top of the underlying network transport protocols, JXTA introduces six P2P protocols. Two of these protocols are referred to as *core protocols* since JXTA requires each and every peer to implement them. The remaining four protocols are called *standard protocols* because their implementation, although being optional, is strongly recommended in order to provide greater interoperability and broader functionality. On top of the JXTA protocols it is possible to develop any kind of P2P services. Some of these services are already provided by the JXTA reference implementation: they are called *standard services*. Other application-specific services, to be developed on top of JXTA, are referred to as *community services*. These services may have their own front-end applications, which are referred to as *community applications*.

5.5.5.2 The Endpoint Routing Protocol

One of the JXTA core protocols is the *Endpoint Routing Protocol* (ERP). The Endpoint Routing Protocol is basically a mechanism to send messages between peers that are not directly connected, or that cannot communicate directly using the same kind of endpoint (i.e. the same network transport protocol). From the perspective of the Endpoint Routing Protocol, every peer has one or more endpoints which represent the available network transport protocols. Each endpoint is identified by an *endpoint address* in the form:

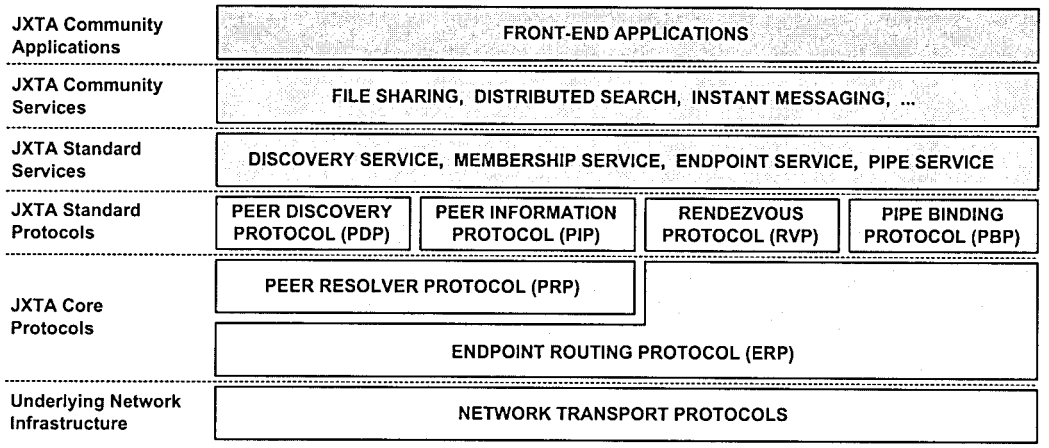


Figure 5.52: The JXTA software architecture

protocol://network_address/service_name/service_parameter

where protocol is the name of the network transport protocol to use with to communicate with the endpoint, network_address is the transport-specific address used to locate the endpoint in the network, service_name (optional) identifies a service at the destination peer, and service_parameter (optional) specifies some parameter to be passed to the service.

The Endpoint Routing Protocol allows peers to discover how to route messages to other peers via intermediaries. In this case, a route is a sequence of endpoints which the message should travel through until it reaches its final destination peer. It should be noted that peers may leave or enter the network spontaneously, so the route between any two given peers may be constantly changing. Whenever a source peer wants to send a message to a destination peer, it must first find a route to the destination peer and then send the message through that route. To support this behavior, the Endpoint Routing Protocol makes use of three kinds of messages: the Route Query Message, the Route Response Message and the Endpoint Router Message.

The use of these messages is illustrated in figure 5.53. First, the source peer sends a Route Query Message to all peers that it is directly connected to (step 1). In this Route Query Message, the source peer specifies the endpoint address of the destination peer it wants to reach. The peers which do not know the route to the desired destination peer must ignore the Route Query Message. The peers which know how the desired destination peer can be reached must reply with a Route Response Message (step 2). This route response contains a list of intermediary endpoints that the message should go through in order to reach the final destination peer. Having received this information, the source peer inserts the route into the original message that it wants to send to the destination peer; the original message becomes an Endpoint Router Message. The source peer sends the Endpoint Router Message to the first endpoint in the route (step 3). When the intermediary peer receives the Endpoint Router Message, it removes its own endpoint address from the route information and sends the message to the second intermediary peer (step 4), and so on until the message reaches its final destination (step 5).

The Route Query Message, Route Response Message and Endpoint Router Message are XML messages whose structure is illustrated in figure 5.54. The Route Query Message has only three elements: the Type element must be set to "RouteQuery", the Destination element contains the endpoint

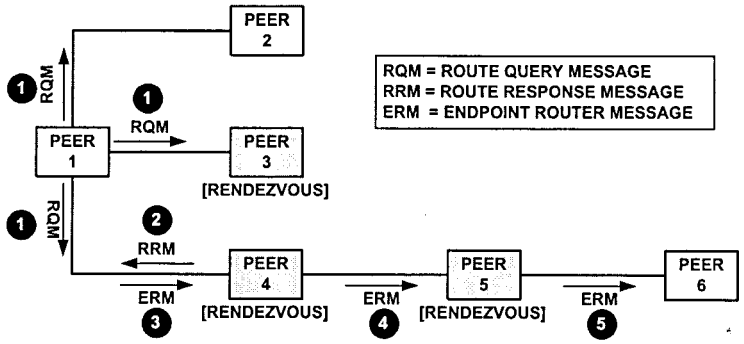


Figure 5.53: Message sequence in the JXTA Endpoint Routing Protocol[97]

address of the final destination peer, and the RoutingPeerAdv element contains the Peer Advertisement of the peer which is requesting the route information.

In the Route Response Message, the Version element must be set to "2", the Type element must be set to "RouteResponse", the Destination element contains the endpoint address of the final destination peer, the RoutingPeer element contains the endpoint address of the peer which is acting as the source of information, the RoutingPeerAdv element contains the Peer Advertisement of the peer which requested the route information, and the NumberOfHops element specifies the number of network hops in the route to the final destination peer. One or more GatewayForward elements specify a series of endpoint addresses, which is the route that messages must follow in order to reach the destination peer.

The message that the source peer wants to send to the destination peer must be embedded as a set of XML elements in an Endpoint Router Message. Besides application-specific elements, the Endpoint Router Message includes the standard elements shown in figure 5.54(c). The Source element contains the endpoint address of the source peer, the Destination element contains the endpoint address of the final destination peer, the Last element contains the endpoint address of the last peer which forwarded the message, and the ReverseHops element specifies the number of network hops in the reverse route to the source peer. One or more GatewayForward elements specify the forward route to the final destination peer, whereas one or more GatewayReverse elements specify the reverse route to the source peer.

When an intermediary peer receives an Endpoint Router Message, it modifies the message before sending it to the next peer. Basically, the intermediary peer removes its own endpoint address from the forward route information, and adds its endpoint address to the reverse route information. The result is that as the message travels through an intermediary peer, the number of GatewayForward elements decreases, whereas the number of GatewayReverse elements increases. Actually, the reverse route information is not mandatory, but JXTA strongly encourages peers to fill in the GatewayReverse elements. At each intermediary peer, the next peer can be determined by means of a specified GatewayForward element or by means of a new route query for the final destination peer.

[97] adapted from [Wilson, 2002]

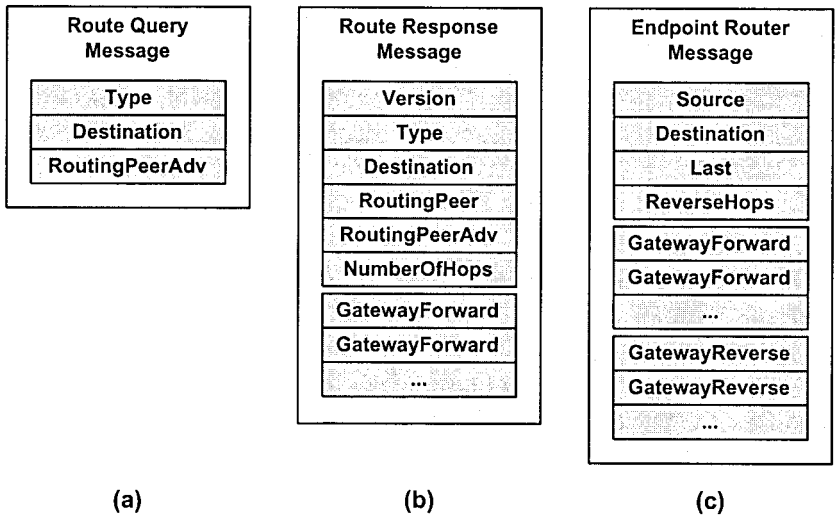


Figure 5.54: Structure of Endpoint Routing Protocol messages

5.5.5.3 The Peer Resolver Protocol

The second JXTA core protocol is the Peer Resolver Protocol (PRP). The Peer Resolver Protocol allows the dissemination of queries to one or multiple peers within a peer group. Its purpose is to provide a generic query/response mechanism that can be used in higher-level services. From the perspective of the Peer Resolver Protocol, each peer is assumed to have one or more *query handlers*, each with its specific semantics. For example, the JXTA reference implementation includes a handler for the Peer Resolver Protocol that responds to queries for certain types of advertisements. According to the Peer Resolver Protocol, each query is addressed to a specific handler name, but not necessarily to a specific peer. Usually, each query is sent to all peers within a peer group, but it can only be handled by those peers which implement the specified handler.

The Peer Resolver Protocol relies on rendezvous peers to disseminate a query to multiple peers, and it relies on endpoint messages to send queries to a specified peer. The Peer Resolver Protocol needs only two kinds of messages: the Resolver Query Message and the Resolver Response Message. In a broadcast query, the source peer sends a Resolver Query Message to all peers that it is directly connected to (step 1). In this Resolver Query Message, the source peer specifies the destination handler's name, as well as a arbitrary query string to be parsed by the destination handlers. Having received the Resolver Query Message, the peers which implement the specified handler must give the message to the handler. Those peers which do not have such a handler must discard the message or, in the case of rendezvous peers, they must forward it to other peers (step 2). When a handler receives a Resolver Query Message it reads the query string, and it may or may not generate a response. If the handler generates a response, it must send it back to the source peer (steps 3 and 4).

The Resolver Query Message and the Resolver Response Message are XML messages whose structure is illustrated in figure 5.56. The Resolver Query Message has five elements: the HandlerName element specifies the name of the handler that the message should be handed over to at the destination peer, the Credential element contains an authentication token that identifies the source peer and authorizes it to send a Resolver Query Message to the peer group, the QueryId element contains a unique identifier that should be sent back in replies to this query, the SourcePeerId element

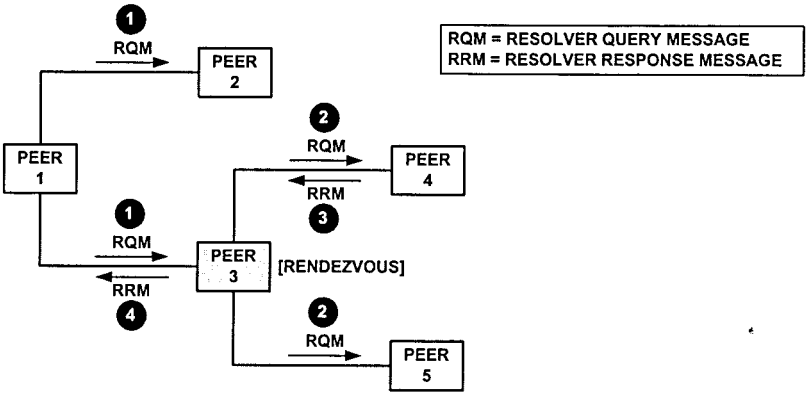


Figure 5.55: Message sequence in the JXTA Peer Resolver Protocol[98]

contains the peer id of the source peer, and the Query element contains an arbitrary query string. Only the destination handler is responsible for parsing and interpreting the query string.

The Resolver Response Message has four elements: the HandlerName element specifies the name of the handler which should receive this response, the Credential element contains an authentication token that identifies the responding peer and authorizes it to send a Resolver Response Message within the peer group, the QueryId element contains the same value as in the original query, and the Response element contains an arbitrary response string. Only the destination handler is responsible for parsing and interpreting the response string.

5.5.5.4 The Rendezvous Protocol

The Rendezvous Protocol (RVP) is one of the four JXTA standard protocols. The Rendezvous Protocol allows a peer to use a rendezvous peer in order to propagate messages to other peers within a peer group. Basically, the Rendezvous Protocol specifies that, before a peer can propagate messages through a rendezvous peer, it must first obtain a *lease* from that rendezvous peer, sometimes referred to also as a *connection lease*. A connection lease is a kind of permit that allows a peer to

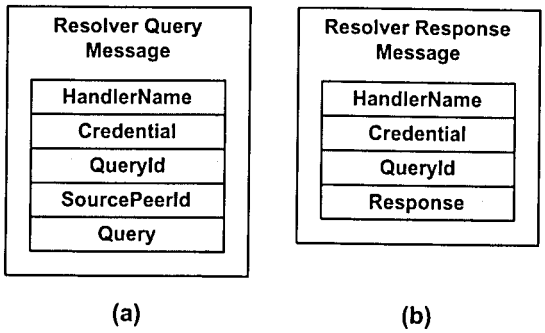


Figure 5.56: Structure of Peer Resolver Protocol messages

[98] adapted from [Wilson, 2002]

use a rendezvous peer for a limited amount of time. After the lease expires, the peer must request a new lease in order to continue using the rendezvous peer to propagate messages. If the peer is no longer interested in using a rendezvous peer it may let the connection lease expire, or it may request the rendezvous peer to cancel a connection lease that is still valid. In either case, the peer will no longer be able to use the rendezvous peer, and it will no longer be able to receive messages from that rendezvous peer.

The Rendezvous protocol makes use of four kinds of messages: the Lease Request Message, the Lease Granted Message, the Rendezvous Propagate Message, and the Lease Cancel Message. Figure 5.57 illustrates the use of the first three of these kinds of messages. First, the source peer that wants to propagate messages via a rendezvous peer sends a Lease Request Message to that rendezvous peer (step 1). In the Lease Request Message the source peer includes its Peer Advertisement, and the rendezvous peer will decide whether to grant a lease to that peer or not. If the rendezvous peer approves the request, it grants a connection lease to the source peer by replying with a Lease Granted Message (step 2). The source peer will be able to use the rendezvous peer for the period of time specified in that connection lease.

Within that time period, the source peer may propagate one or more Rendezvous Propagate Messages via the rendezvous peer. Each Rendezvous Propagate Message is addressed to a specific service, rather than being address to a specific peer. Only those peers which implement that particular service are required to understand the service-specific elements of the message. On the other hand, all peers should make their best effort to propagate the message to other peers. A rendezvous peer must forward the message to all peers it knows about using endpoint communication (steps 4 and 5). In addition, if the network transport protocols allow it, all peers (even simple peers) should propagate the message using transport-specific broadcast mechanisms; with TCP/IP, each peer propagates the message using TCP multicast (step 6). As a result, if a peer is located in the same LAN segment as the rendezvous peer, it will receive the Rendezvous Propagate Message twice: once via endpoint communication and once via TCP multicast.

The Lease Request Message, the Lease Granted Message, the Lease Cancel Message, and the Rendezvous Propagate Message are XML messages whose structure is illustrated in figure 5.58. The

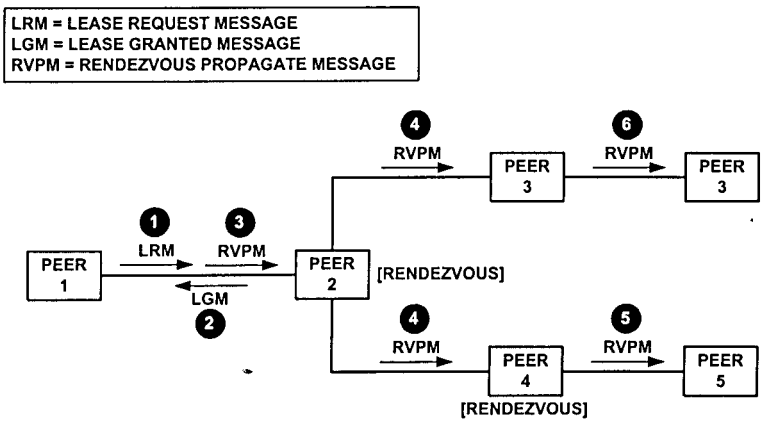


Figure 5.57: Message sequence in the JXTA Rendezvous Protocol[99]

[99]adapted from [Wilson, 2002]

Lease Request Message and the Lease Cancel Message have a single element, which contains the Peer Advertisement of the peer requesting the lease or canceling the lease, respectively. In the Lease Granted Message, the RdvAdvReply element contains the Peer Advertisement of the rendezvous peer, the ConnectedPeer element contains the peer id of the rendezvous peer, and the ConnectedLease element is a string representation of the lease time in milliseconds. It should be noted that, whereas the Lease Granted Message is the response to a Lease Request Message, there is no response message to a Lease Cancel Message.

The elements of a Rendezvous Propagate Message are shown in figure 5.58(d). The MessageId element contains a unique identifier for the message, the DestSName element contains the name of the destination service for the propagated message, and the DestSParam element contains the parameters for the destination service. The TTL element specifies the maximum number of network hops for message propagation; this value is decreased at each peer and once it reaches zero the Rendezvous Propagate Message should not be propagated anymore. One or more Path elements contain the peer ids of peers which the Rendezvous Propagate Message has already been through; a rendezvous peer will propagate the message to all peers it knows about except for: (1) the message sender and (2) the peers specified in these Path elements.

5.5.5.5 The Peer Discovery Protocol

A peer may learn of the existence of rendezvous peers by means of their published advertisements. And it may learn of the existence of other resources, such as peer groups, services and pipes, by means of their corresponding advertisements. The Peer Discovery Protocol (PDP) is the JXTA standard protocol which enables a peer to find published advertisements within its peer group. The Peer Discovery Protocol defines two kinds of messages: the Discovery Query Message and the Discovery Response Message. However, the Peer Discovery Protocol is not responsible for propagating these messages; instead, it relies on the Peer Resolver Protocol to send and receive the discovery messages. The Discovery Query Message and the Discovery Response Message can be encapsulated in the Query and Response elements shown in figures 5.56(a) and 5.56(b), respectively.

The Discovery Query Message and the Discovery Response Message are XML messages whose structure is illustrated in figure 5.59. In both kinds of message, the Type element is a numeric value that identifies the desired type of advertisement: "0" refers to Peer Advertisements, "1" refers to Peer

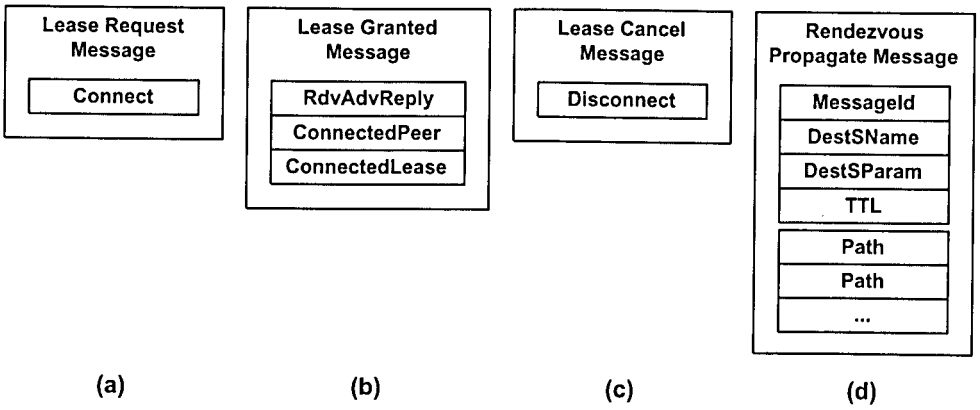


Figure 5.58: Structure of Rendezvous Protocol messages

Group Advertisements, and "2" refers to any other type of advertisement. In the Discovery Query Message, the Threshold element specifies the maximum number of advertisements that should be sent by a peer responding to the query, and the PeerAdv element contains the Peer Advertisement of the peer making the query. The Attr and Value elements specify the search criteria: Attr contains the name of an element in the desired advertisements, and Value specifies the desired value for that element. Only those advertisements which have the specified element with the specified value should be returned in response to the query.

In the Discovery Response Message, the Count element specifies the total number of response elements in the message, and the PeerAdv element contains the Peer Advertisement of the peer that is responding to the query. The Attr element and the Value element must be the same as those in the original query. One or more Response elements contain the advertisements that match the search criteria. Each of the Response elements has an Expiration attribute, which is not shown in figure 5.59(b), but which specifies the period of time during which the returned advertisement should be considered valid.

A peer may cache advertisements discovered via the Peer Discovery Protocol for later usage. The ability to cache advertisements is not required by JXTA, but it is an important optimization that can significantly increase performance: caching avoids having to perform a new discovery query each time a peer wants to access an advertisement. However, cached data cannot be made permanent since a P2P network is a dynamic environment where peers may appear, disappear or migrate at any time without notice. When a peer caches an advertisement, it becomes a source for that advertisement: if a discovery query passes through this peer, the peer may return its local copy instead of going to the original source. Caching is especially appropriate for rendezvous peers since they receive most of the discovery queries traveling across the P2P network.

5.5.5.6 The Peer Information Protocol

The Peer Information Protocol (PIP) is another JXTA standard protocol. The purpose of the Peer Information Protocol is to allow peers to obtain status information from previously discovered peers. Currently, this status information is limited to the uptime of peers and the amount of traffic handled by their endpoints. The original purpose of the Peer Information Protocol is to allow peers to monitor

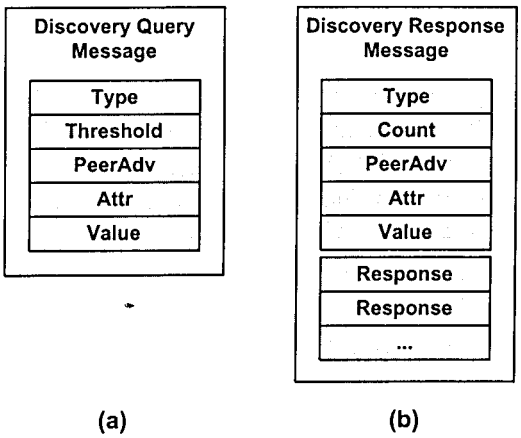


Figure 5.59: Structure of Peer Discovery Protocol messages

the status of one another so that they can make optimal decisions about how they can use each other. The ultimate goal is to make the most efficient use of resources in a P2P network. For example, a peer could use status information to decide which rendezvous peers it should connect to, by avoiding the rendezvous peers with heavier load.

The Peer Information Protocol requires two kinds of messages: the Peer Info Query Message and the Peer Info Response Message. As in the Peer Discovery Protocol, the Peer Information Protocol also relies on the Peer Resolver Protocol to send and receive status information to and from other peers. However, contrary to the Peer Discovery Protocol, each Peer Info Query Message has a specific destination peer given by the target peer's id, although it may be propagated to many other peers. When a peer receives a Peer Info Query Message, it must check whether the target peer id matches its own peer id or not. If it does, the peer generates a Peer Info Response Message; otherwise, the peer propagates the message to other peers.

The Peer Info Query Message and the Peer Info Response Message are XML messages whose structure is illustrated in figure 5.60. In both kinds of message, the SourcePId element contains the peer id of the peer requesting the status information, and the TargetPId element contains the peer id of the peer from which status information is being solicited. In the Peer Info Query Message, the Request element contains an arbitrary string that specifies the status information being requested; in the Peer Info Response Message, the Response element contains an arbitrary string that specifies the status information being returned. Currently, the status information can only be the uptime, the network traffic, or both; in the future, additional status information may be considered.

The UpTime element specifies the amount of time, in milliseconds, since the peer joined the P2P network. The TimeStamp element contains the time, up to millisecond precision, when the target peer generated the status information being returned. The Traffic element contains details about the network traffic handled by the target peer. The LastIncomingMessageAt element specifies the last time that one of the target peer's endpoints handled an incoming message, and the LastOutgoingMessageAt element specifies the last time that one of the target peer's endpoints handled an outgoing message. The In and Out elements describe the amount of inbound and outbound network traffic, respectively, handled by each of the target peer's endpoints. Each Transport element contains the

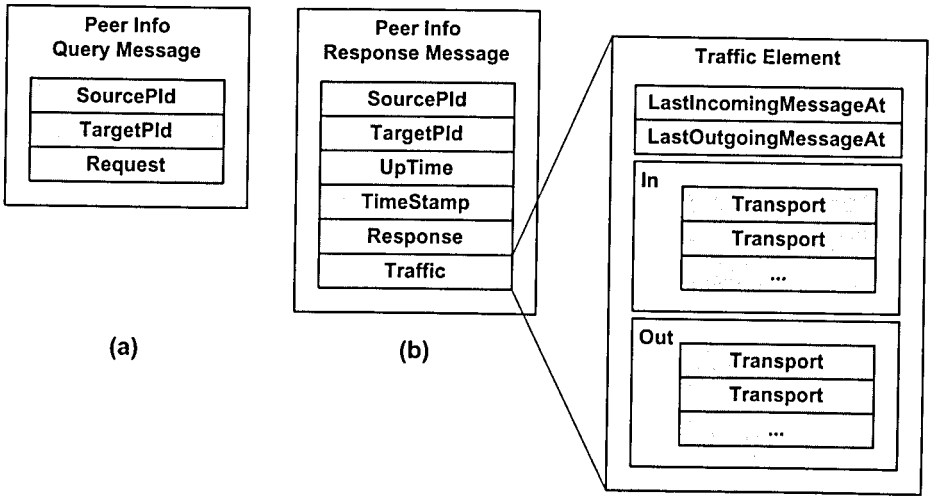


Figure 5.60: Structure of Peer Information Protocol messages

number of bytes that each endpoint has sent (if inside the Out element) or received (if inside the In element).

5.5.5.7 The Pipe Binding Protocol

Within a peer group, a source peer can send messages to one or more target peers using either the Endpoint Routing Protocol presented in section 5.5.5.2, or the Pipe Binding Protocol (PBP), which is the fourth JXTA standard protocol. In fact, the Pipe Binding Protocol is built on top of the Endpoint Routing Protocol. Typically, peers will use pipes to communicate with services provided by other peers in the same peer group. The advantage of using pipes is that peers can easily establish a virtual communication channel between their endpoints, without having to worry about how the Endpoint Routing Protocol will route messages between the endpoints. A pipe is a unidirectional communication channel, so it allows one peer to send messages and one or more peers to receive them; for the sending peer the pipe is an output pipe, whereas for the receiving peers it is an input pipe.

A pipe is usually described by its corresponding Pipe Advertisement. For a peer to create an output pipe or an input pipe, it must first have access to the Pipe Advertisement. Since a pipe usually connects a peer to a service implemented by another peer, the first peer will be referred to as the service client and the second one as the service provider. In most cases, the service provider already has access to the Pipe Advertisement; in fact, it is the service provider who publishes the Service Advertisement, which contains the Pipe Advertisement required to communicate with the service. Having created the Pipe Advertisement, the service provider can immediately create an input pipe in order to start listening for service requests. If the service is provided by several peers, all of them will have to create an input pipe from the Pipe Advertisement; in this case, the pipe is called a propagate pipe.

A Pipe Advertisement is an XML document, or a portion of an XML document if it is inserted in a Service Advertisement. As shown in figure 5.61(a), a Pipe Advertisement has three elements. The Id element contains a unique pipe identifier, and the Type element specifies whether the pipe is a unicast pipe, a propagate pipe, or a secure unicast pipe. If the Pipe Advertisement is published as a standalone advertisement rather than being part of a Service Advertisement, the Name element contains a symbolic name that can be used in order to discover the Pipe Advertisement via the Peer Discovery Protocol. It should be noted that a Pipe Advertisement does not specify any of its endpoints; instead, a Pipe Advertisement can be used to create a pipe between any two peers (or between more than two peers in the case of propagate pipes). For this reason, peers that create the input pipe need the Pipe Advertisement only, whereas the peer creating the output pipe must specify the input pipes that the output pipe should connect to.

The service client must create an output pipe to communicate with the service, so it must somehow have access to the Pipe Advertisement. This is usually achieved via the Peer Discovery Protocol, which allows the service client to discover the Service Advertisement. From the Service Advertisement, the service client extracts the Pipe Advertisement. Once with the Pipe Advertisement, the service client will be able to create the output pipe. However, having the Pipe Advertisement is not enough to create the output pipe. The service client must also determine which peers have created the corresponding input pipe, so that it can connect its output pipe to those input pipes. This is where the Pipe Binding Protocol comes in.

The Pipe Binding Protocol defines two kinds of messages: the Pipe Binding Query Message and the Pipe Binding Answer Message. First, the service client propagates a Pipe Binding Query Message to all peers it is directly connected to. In this Pipe Binding Query Message, the service

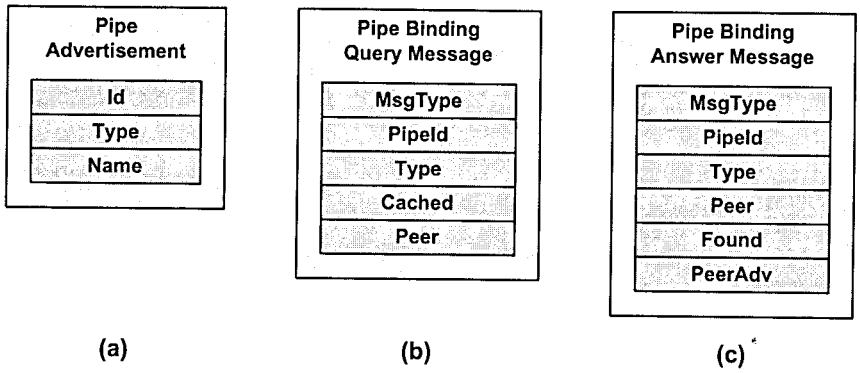


Figure 5.61: Structure of pipe advertisements and Pipe Binding Protocol messages

client includes the original pipe identifier from the Pipe Advertisement. When a peer receives a Pipe Binding Query Message, it checks if it has any input pipe with the specified identifier. If it does, the peer responds with a Pipe Binding Answer Message containing its own Peer Advertisement. For each Pipe Binding Answer Message received, the service client extracts the endpoint information from the Peer Advertisement and binds its output pipe to that endpoint, which represents an input pipe at a service provider.

The Pipe Binding Query Message and the Pipe Binding Answer Message are XML messages whose structure is illustrated in figures 5.61(a) and 5.61(b), respectively. The Pipe Binding Protocol uses the MsgType element to distinguish between the two messages: in the Pipe Binding Query Message this element is set to "Query", whereas in the Pipe Binding Answer Message it is set to "Answer". In both messages, the PipeId element contains the same pipe identifier as in the original Pipe Advertisement, and the Type element specifies the same kind of pipe as in the original Pipe Advertisement.

In the Pipe Binding Query Message, the Cached element specifies whether the remote peers being queried can look up the input pipe in their cache of previously bound input pipes, or if they have to actually check if the requested input pipe is still active. The Peer element may contain a peer id of the only peer that should respond to the query; this element is only used when the service client already knows which peer has the desired input pipe. In this case, the Pipe Binding Answer Message must contain an identical Peer element, and the Found element indicates whether an input pipe with the specified pipe identifier was found at that peer or not. The most important element in the Pipe Binding Answer Message is the PeerAdv element, which contains the Peer Advertisement of the peer that has an input pipe with the original pipe identifier.

5.5.5.8 Standard services

The JXTA reference implementation provides a complete implementation of the six JXTA protocols. In addition, it provides a set of standard services which encapsulate these protocols and simplify their use. The JXTA reference implementation has been developed in Java, but it could have been developed using any other programming language as well. The JXTA standard services provide a higher-level API that is closer to the programming model that application developers are already familiar with, rather than to the particular details of JXTA protocols. In fact, the standard services

hide most of the details concerning protocol-specific messages and their exchange between peers. The standard services that the JXTA reference implementation provides are the following:

- The *EndpointService* encapsulates the Endpoint Routing Protocol. The EndpointService allows a peer to send messages to other peers' endpoints without having to worry about finding routes to those endpoints. In addition, it allows peers to receive endpoint messages and handle them in any application-specific way. For this purpose, each peer may implement the EndpointListener interface in order to be notified of messages received by the EndpointService. The EndpointService also allows a peer to retrieve information about its own endpoints, and to check whether a given remote endpoint address is reachable before sending an endpoint message.

- The *ResolverService* encapsulates the Peer Resolver Protocol, and it provides a generic mechanism for peers to send queries and receive responses. The syntax and semantics of these queries and responses is application-specific. The ResolverService allows peers to register their own query handlers, each having a different name. The ResolverService will pass each received message on to the handler specified in the HandlerName element, as shown in figure 5.56.

- The *RendezVousService* encapsulates the Rendezvous Protocol, and it is responsible for propagating messages within a peer group. The RendezVousService defines a subscribe mechanism that allows simple peers to receive propagated messages, and that allows rendezvous peers to receive and automatically forward messages to other peers. The RendezVousService also allows a peer to connect to or disconnect from a rendezvous peer without having to worry about requesting, renewing or canceling connection leases.

- The *DiscoveryService* encapsulates the Peer Discovery Protocol, and it provides an asynchronous mechanism for discovering advertisements. The DiscoveryService allows a peer to query other remote peers for advertisements, without having to worry about building discovery queries and parsing returned responses. Each peer may implement the DiscoveryListener interface in order to be notified of all responses that the DiscoveryService receives. The DiscoveryService also allows a peer to publish its own advertisements either locally or remotely at other peers if they support caching.

- The *PeerInfoService* encapsulates the Peer Information Protocol, and it provides an asynchronous mechanism for retrieving status information from a remote peer. The PeerInfoService also allows a peer to retrieve its own status information. According to the PeerInfoService, a peer may implement the PeerInfoListener interface, which will be invoked when a response is received from the remote peer.

- The *PipeService* encapsulates the Pipe Binding Protocol, and it allows peers to create input and output pipes based on a given Pipe Advertisement. Typically, this Pipe Advertisement can be published and discovered using the DiscoveryService. The PipeService is able to create an output pipe and bind it to local and remote endpoints, taking care of querying other peers for matching input pipes. The service client may specify the maximum time this operation should take, or it may implement the OutputPipeListener interface in order to be notified as soon as the pipe binding operation is completed. Each peer may implement the PipeMsgListener interface, which is invoked when a new pipe message is received.

- The *MembershipService* does not encapsulate any particular JXTA protocol. Its purpose is to allow a peer to join a peer group and to establish its own identity within that peer group.

Each peer group has its own MembershipService, which is responsible for authenticating peers before allowing them to enter the group. First, the peer provides a credential, and the MembershipService returns an authentication object that encapsulates the credential and specifies the authentication method that should be used with that credential. The possible authentication methods, as well as how those methods are applied to credentials, are specific features of each peer group. After the credential has been successfully authenticated, the peer can request the MembershipService to join the group. The MembershipService is also be invoked when the peer wants to leave the group.

5.5.5.9 Community services

The JXTA protocols and standard services establish a P2P platform on top of which higher-level P2P services, such as file sharing and instant messaging, can be implemented. The P2P services built on top of JXTA are referred to as JXTA community services. The reason for this designation is that JXTA was conceived as an open-source project that promotes the development of P2P solutions based on the JXTA platform; project JXTA hosts these new projects and encourages them to be developed as open-source solutions as well, so that they become available to the wider JXTA community. The JXTA community services can be developed as a set of one or more *modules*. The concept of module has been introduced in order to allow application developers to build functionality that extends the original JXTA platform in a modular fashion.

In practice, a module is any piece of functionality that a peer is able to run and, just like any piece of software, a module has a specification and an implementation. However, JXTA realizes that each module specification may have several implementations, for example using alternative technologies. In addition, JXTA realizes that each module will likely be improved or changed over time, leading to increasingly sophisticated specifications and implementations. For these reasons, JXTA has divided module advertisements into two separate advertisements: the Module Specification Advertisement and the Module Implementation Advertisement.

The term "Service Advertisement" used in the previous sections actually refers to the Module Specification Advertisement which, as shown in figure 5.62(b), may contain a Pipe Advertisement to communicate with the service (i.e., the module implementation).

JXTA also realizes that there classes of module functionality, regardless of its particular specification or implementation. For example, a given module may belong to a class of modules providing file sharing functionality, or it may belong to a class of modules providing instant messaging capabilities. Classes of modules are represented by their own advertisement, which is called the Module Class Advertisement.

The Module Class Advertisement, the Module Specification Advertisement and the Module Implementation Advertisement are XML documents whose structure is illustrated in figure 5.62. The Module Class Advertisement has three elements: the MCID element contains a unique identifier for the module class, and the Name and Desc elements contain a name and description for that module class.

In the Module Specification Advertisement, the MSID element contains a unique identifier for the module specification, and the Name and Desc elements contain a name and description for that module specification. The Vers element specifies the version of the module specification being advertised, the Crtr element identifies the author of the specification, and the SURI element contains a URI to download the specification document. The PipeAdvertisement element contains an embedded Pipe Advertisement that the module implementation uses to create an input pipe; other peers that want

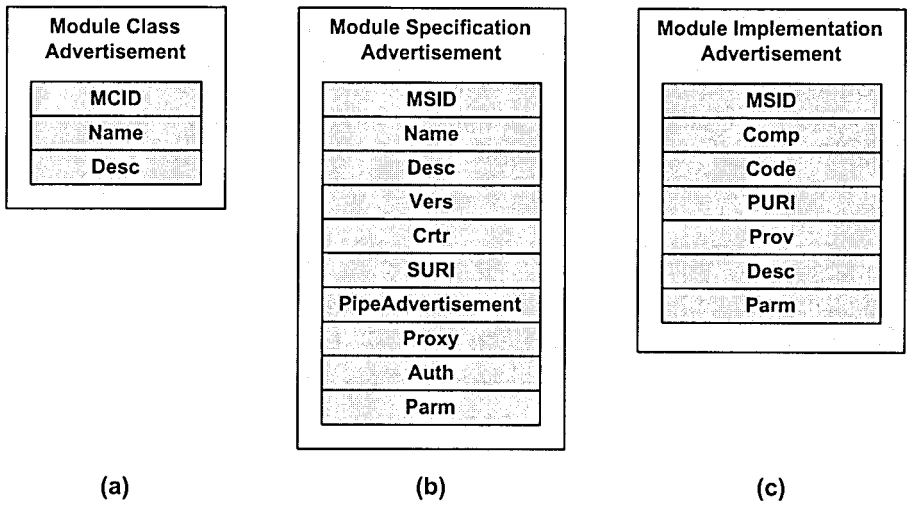


Figure 5.62: Module advertisements

to interact with the module must create an output pipe based on this same Pipe Advertisement. The Proxy element may contain the identifier of another module specification, which describes a proxy module used to communicate with the module of this specification. Actually, JXTA discourages the use of proxy modules, and the same applies to the Auth element, which identifies an external authentication module. The Parm element may contain any application-specific parameters.

In the Module Implementation Advertisement, the MSID element contains the same identifier as the corresponding Module Specification Advertisement. The Comp element (where "Comp" stands for "compatibility") describes the environment in which the module implementation may be executed, and the Code element contains any information required to run the module implementation. The PURI element contains a URI that can be used to download the module implementation. The Prov element identifies the entity which provides the module implementation, the Desc element contains a description of the module implementation, and the Parm element may contain any application-specific parameters.

JXTA does not require peers to publish all the three advertisements for a given module. Typically, a peer will create a Module Specification Advertisement to describe the module, and a Pipe Advertisement to allow other peers to communicate with the module implementation. The peer will embed the Pipe Advertisement into the Module Specification Advertisement and only then will it publish the resulting Module Specification Advertisement. Other peers that wish to interact with the module can use the DiscoveryService to find the Module Specification Advertisement with a specific module name. The peer may also publish the Module Class Advertisement, if it is not already available. As for the Module Implementation Advertisement, it is mainly useful in order to disseminate a module implementation throughout the P2P network, so that other peers can download it and run it themselves; this allows peers to introduce new services in the P2P network, as well as the module implementations that support those services.

5.5.5.10 Security in JXTA

In traditional client/server systems, security measures are usually implemented by means of centralized services; but in a P2P network it may not be appropriate to require a large number of peers to rely on centralized services provided by a third party. Despite this fact, project JXTA still adopts a security model that relies on existing technologies, rather than trying to come up with a new security model for P2P networks.

Before a peer connects to the JXTA network for the first time, it defines a username and a password. Based on these elements, JXTA creates a private key and a public key certificate for that peer. The public key certificate is inserted in the Peer Advertisement, which JXTA publishes as soon as the peer connects to the network. The peer will use the same certificate whenever it connects to the JXTA network again. The private key is stored locally and it is protected by means of the password that only the peer knows. Whenever a peer wants to send a secret to another peer, the first peer can use the public key of the second peer to encrypt the data; then the data can only be decrypted using the corresponding private key, which is held by the second peer.

JXTA makes use of public key certificates in such a way that centralized Certification Authorities (CAs) are not mandatory, but not excluded either. JXTA allows peers to become their own CAs so that they can generate their own certificates. However, peers still have to recognize a common CA when they authenticate one another. This requirement is currently not addressed by JXTA, but it could be implemented by designating one of the members of a peer group as the CA for other members within that group, for example. This approach could be used not only to authenticate users inside the group, but also to authenticate users that want to join the group.

The use of the Transport Layer Security (TLS) protocol allows peers to exchange information over secure communication channels. JXTA provides peers with the possibility of using secure unicast pipes, which are based on TLS. TLS has two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol encrypts data using symmetric keys. These symmetric keys are generated for each connection and are based on a secret negotiated by another protocol, which is usually the TLS Handshake Protocol. JXTA makes an efficient use of TLS by making multiple pipes between two peers always share the same TLS connection.

5.6 Concluding remarks

This chapter has presented an overview of the most relevant technologies when considering the design and implementation of an integration infrastructure. In this work, the purpose of such an infrastructure is to enable applications residing at different enterprises to interoperate with each other. Each technology presented in this chapter has a different approach towards this kind of interoperability. MOM systems establish a single interface that enables each application to exchange messages with any other. B2B frameworks focus on fixed interaction models and standard document formats that allow applications at different enterprises to understand each other. Agent-based systems propose architectures in which autonomous, intelligent, and even mobile agents interact with remote services and with other agents, while acting on behalf of an enterprise or a human user. Web Services is a recent approach that defines how applications available on the World Wide Web can be described and registered, so that other applications can discover them and interact with them. Peer-to-peer networking is a new paradigm that allows applications to interact by means of decentralized services. These approaches are represented in figure 5.63 from (a) to (e), respectively.

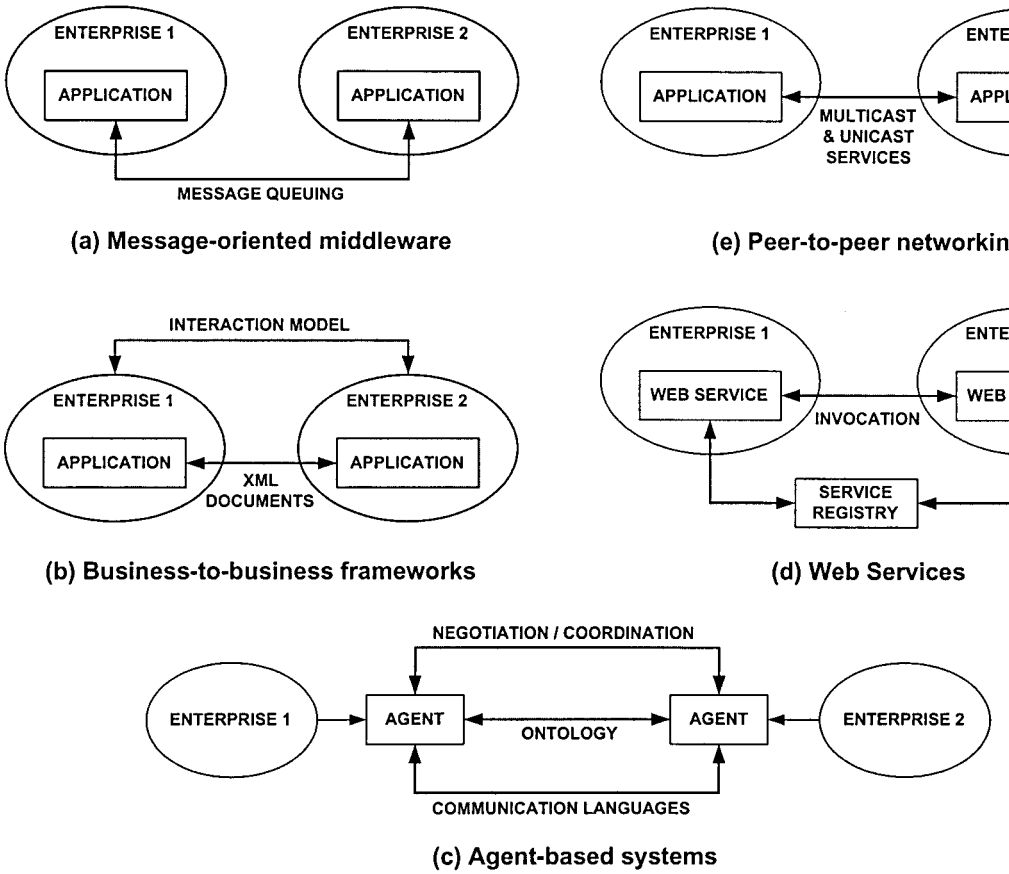


Figure 5.63: Technological options for B2B interoperability

Because of these sharp differences, these technologies cannot be compared directly to each other. For example, it makes no sense to compare agent-based systems with B2B frameworks, or MOM systems to Web services. Each of these technologies has its own focus, and its scope of applicability. In fact, these technologies seem to complement each other in many ways, rather than being alternative solutions. For instance, document formats defined by B2B frameworks could be used as message formats in a MOM system, and Web services would allow intelligent agents to automatically discover and interact with remote applications. Hence, these technologies are useful in different respects, and employing one of them does not necessarily preclude the use of another. A better way to consider these technologies within a single conceptual framework is to regard them as being positioned at different levels, as shown in figure 5.64. At the bottom level, MOM systems enable message exchange between applications. At a higher level, Web services specify each operation as a message sequence. Alternatively, peer-to-peer networking could provide this functionality in a more decentralized way, if appropriate services are developed. Then B2B frameworks define the message formats and sequences of operations across several applications. At the highest level, agent-based systems introduce autonomous agents that travel through the network, interact with applications, and reason about the results obtained.

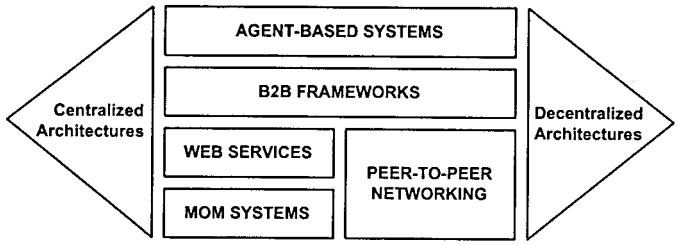


Figure 5.64: Conceptual framework for different technological options

From this it must be concluded that the technologies presented in this chapter have different requirements. A MOM system should be implemented with a middleware technology such as CORBA; Web services require a central service registry; B2B frameworks require messaging facilities and common registries; and agent-based systems are by far the most challenging, requiring knowledge representation, ontologies, reasoning abilities, communication languages, negotiation capabilities, coordination mechanisms, and code mobility. This means that even when these technologies are regarded as being positioned according to increasing levels of abstraction, there is still a substantial gap between agent-based systems and the remaining technologies. It seems that advances in several areas are still required before agent-based systems can demonstrate their full potential. On the other hand, it is clear that agent-based systems are a paradigm of its own kind, comparable in significance not to the technologies presented in this chapter, but perhaps only to workflow management itself.

The relationship between workflow management and agent-based systems is a promising relationship. Much like previous chapters, which have been showing that the combination of two parallel trends can bring significant benefits, so too the combination of workflow management and agent-based systems is likely to bring about remarkable improvements to both paradigms. On one hand, workflow management is cited as being the most powerful technique for agent coordination [Griss and Pour, 2001]; for example, in comparison with six other coordination models, workflow management is top-ranked in every respect, except for autonomy and scalability [Tolksdorf, 2000]. On the other hand, agent-based systems are able to surpass some of the limitations often associated with workflow management systems. To begin with, agents can improve the scalability of workflow management systems. Furthermore, due to their properties, agents can be better equipped to deal with unexpected conditions or exceptional behavior, something that workflow management systems are not very good at. Additional advantages have been identified in section 5.3.6.2.

Sections 5.3.6.1 and 5.3.6.2 have shown that two main relationships between agent-based systems and workflow management should be distinguished. One is that of workflow management systems which control the execution of processes where activities are performed by software agents, or where the ultimate resources are represented by software agents. The other relationship is that of agents which have workflow management capabilities in order to control their internal execution steps, or to control their trip throughout the network in the case of mobile agents. In one way or another, or as a combination of both relationships, software agents and workflow management are becoming the two main ingredients in integration infrastructures for virtual enterprises, as suggested in section 5.3.6.3. For example, the COSMOS architecture presented earlier in figure 3.31 on page 128 includes an agent environment in which mobile agents carry contracts between enterprises as many times as necessary until contracts are signed. However, mobile agents raise several security concerns [Farmer

et al., 1996], some of which have been discussed in section 5.3.1.1. These security concerns are the primary obstacle to adopting mobile agents in business environments.

As section 5.3.6 has shown, there are currently many efforts towards integration architectures that combine software agents and workflow management capabilities. At the same time, it is possible to make use of workflow management capabilities in combination with other technologies as well. For example, by reshaping its FlowMark product into the successful MQSeries Workflow product, IBM has proved that MOM systems are a suitable infrastructure for workflow management systems. On the other hand, the development of the Business Process Execution Language for Web Services (BPEL4WS) is an indication that the promoters of Web Services are becoming aware of the opportunities for workflow management within this recent technology trend. Regarding B2B frameworks, only RosettaNet has made explicit use of workflow management principles by defining some of its Partner Interface Processes (PIPs) as workflow processes. However, B2B frameworks can become more flexible and widen its range of applicability if they are combined with workflow management capabilities [D. Ferreira and J. Ferreira, 2001].

B2B frameworks focus on architectures for the exchange of information between business partners, with a special emphasis on XML as the enabling technology for defining and exchanging business documents. Typical B2B frameworks comprise, in general, two main approaches for tackling interoperability. On one hand, they specify a set of common data formats that allow the exchange of information between trading partners within the same industry or service sector. On the other hand, they rely on particular models of B2B interaction, which define how document exchanges take place between several role players. For example, OBI defines the order documents to be exchanged between trading partners according to the interaction model shown in figure 5.13 on page 185; bolero.net introduced boleroXML in order to specify the documents to be exchanged during the settlement process described in section 5.2.4.3; and RosettaNet proposes common dictionaries that define the product and business terms to be used in PIPs.

Other frameworks, particularly ebXML, propose generic architectures that allow the target companies to define their own document formats and interaction models. It should be noted, however, that ebXML is more akin to the inter-enterprise workflow architectures described earlier in section 3.5.3 on page 118 than to other B2B frameworks. The ebXML registry, which stores business process models, is similar in purpose to the WISE online catalog (section 3.5.3.6) or even to the ACE-Flow service catalog (section 3.5.3.4); and the Collaboration Protocol Agreement (CPA), that ebXML proposes as the result of two matching Collaboration Protocol Profiles (CPPs), plays the same role as the electronic contract in CrossFlow (section 3.5.3.8), which results from a match between a Contract Search Template (CST) and a Contract Advertising Template (CAT).

As discussed in chapter 2, the Internet and its associated technologies have had and are still having a great impact on business practices. Therefore, B2B frameworks, which can be regarded as standardization efforts in B2B e-commerce, face a wide diversity in the kind of B2B interactions that take place over the Internet. In order to keep their standardization effort within a manageable scope, B2B frameworks have put their focus on particular business documents or interaction models. For example, the cXML framework addresses procurement activities alone, regardless of industry sector. On the other hand, the RosettaNet standards apply mainly to the Information Technology and Electronic Components sectors, while supporting an extensive set of processes ranging from marketing to inventory and order management. These approaches actually hamper the applicability of B2B frameworks [D. Ferreira and J. Ferreira, 2001].

A narrow focus on a particular industry sector or on particular business processes confines the B2B framework to a limited use and precludes the implementation of the same framework in other

business environments. In fact, the RosettaNet framework could be applied to the pharmaceutical industry or to the automotive industry, for example, if the appropriate dictionaries and PIPs were specified; and the approaches proposed by OBI and cXML could be applied to business activities other than procurement. However, by assuming a particular B2B interaction model, the whole framework becomes dependent upon that model and it can be hardly employed if a different B2B interaction model is to be considered. Hence, changes in the interaction model are not allowed. In addition, alternative solutions will have to be chosen or worked out for other B2B scenarios, increasing the number of different - and therefore incompatible - solutions being implemented.

There is certainly an interaction model for every B2B scenario, but no particular model should be taken for granted. The design of a B2B integration infrastructure must leave room for subsequently defining the interaction model (in effect, the business process) that the trading partners will perform. But then, supporting the definition and execution of business processes over an integration infrastructure is precisely the purpose of workflow management systems. Thus, B2B frameworks could be combined with workflow management capabilities in order to become more flexible, and applicable to a wider range of B2B scenarios. In conclusion, workflow management will prove to be advantageous when combined with B2B frameworks, besides agent-based systems, MOM systems, or Web Services.

However, when considering the choice of an infrastructure to support inter-enterprise workflow management, one should immediately realize that the higher the chosen technology is on the conceptual framework shown in figure 5.64, the more challenging it will be to implement a B2B integration infrastructure based on that technology, because each of these technologies brings challenges of its own. First, Web Services represent a recent technology, and a lot of standardization is still taking place in this field; not only more standards are required but, at the same time, the core standards are being significantly improved. Second, B2B frameworks focus mainly on the standardization of business documents and, in general, they apply to particular B2B scenarios only; to support the exchange of business documents, B2B frameworks rely either on Internet protocols or on special-purpose messaging platforms. Third, agent-based systems, as previously concluded, have exceedingly demanding requirements, they are vulnerable to security issues, and they are likely to become more attractive only in light of future advances.

This leaves MOM systems, which provide essential messaging capabilities, as the preferred choice. The next chapter describes two research projects which have produced two workflow management systems based on MOM platforms. However, the next chapter will eventually confirm that, to support the engineering of business networks, a more decentralized platform is required. In this context, peer-to-peer networking will become an important technology, as long as it is possible to provide an appropriate set of business-supporting services on top of such platform, as will be proposed in chapter 7.

Chapter 6

Experiments and Findings

According to the conclusions drawn in chapters 3 and 4 every workflow management system can be regarded, and it should be effectively designed, as a compound of two distinct facilities. On one hand there is what could be referred to as a *workflow facility*, which provides process modeling and execution capabilities. On the other hand there is an integration infrastructure, or *integration backbone*, that enables information exchange between all resources and between the workflow facility and those resources. Two experiments have been absolutely essential in establishing such a workflow management system architecture, and in proving that the workflow facility and the integration backbone can indeed be developed as separate components, employing different technologies if necessary.

These experiments were the two research projects PRONEGI and DAMASCOS. The first of these projects, PRONEGI, has developed a system architecture based on workflow management and software components in order to support Total Quality Management (TQM) [Oakland, 1993]. Despite its focus on internal business processes, some features of the PRONEGI system architecture can be useful in inter-enterprise scenarios as well. The DAMASCOS research project, which builds partly on results from PRONEGI, has developed a workflow-enabled messaging infrastructure to integrate several software applications residing both at the enterprise and at its business partners. The DAMASCOS project has faced important issues concerning the extension of its integration infrastructure across several enterprises.

6.1 The PRONEGI project

The PRONEGI project (Integration Support to Business Processes Applied to Total Quality Management) [Macedo and Ferreira, 1998] was a national research project established between INESC Porto and SONAFI (Sociedade Nacional de Fundição Injectada S.A.). SONAFI is a Portuguese company specialized in the manufacturing of small- and medium-sized metallic pieces for the automotive industry. SONAFI employs sophisticated machining techniques based on pressure die-casting[100] in aluminum and zinc alloys; some of the resulting products are shown in figure 6.1. Besides manufacturing these parts, SONAFI also manufactures tools for its own internal use. SONAFI finishes castings according to customer requirements, employing several methods such as *shotblasting* and *vibro-deburring*[101]. In fact, and in order to comply with increasingly demanding customer require-

[100] The following site provides a short explanation of what die-casting is: http://www.diecasting.org/faq/introduction/whatis.htm

[101] Both of these are methods of finishing metal surfaces.

ments, SONAFI has been improving its production processes, especially regarding quality standards. Since 1994, SONAFI has been awarded several quality certifications including ISO 9002, QS9000, ISO TS 16 949, and ISO 14001.

By 1998, SONAFI already had an information system supporting TQM called the Quality Toolkit, which was a client-server, database-centered system developed by INESC Porto. However, the purpose of the PRONEGI project was to develop an information system architecture founded on business process modeling, so that this new system could be adapted to the organization throughout its evolution. Two of the main goals of PRONEGI were to build a functional and informational reference model for software applications supporting total quality management (TQM), and to identify a library of *functional operations* that would allow the execution of workflow processes. Any software application supporting TQM processes could then be assembled from these functional operations, which would be brought under workflow control, rather than being developed as a monolithic system such as the Quality Toolkit. The new system should be able to cope with the evolution of the company's business processes simply by reconfiguring the workflow models and developing any additional functional operations deemed necessary.

The PRONEGI project argues that current systems development methodologies jump directly from business models, which express user requirements, into software applications that attempt to meet those requirements. This results in monolithic, inflexible information systems that are unable to adapt to evolving requirements. Thus, there is a gap between those two stages and that gap must be filled by converting high-level business process models into executable process models. These executable models will be described as sequences of activities, where each activity makes use of a certain system functionality. The final information system is the compound of the whole functionality required to support those activities.

This approach is built on two main assumptions. The first assumption is that the business processes must be carefully examined in order to identify not only their activities, but also generic activities that can take place in one or more business processes; these will be referred to as *partial activity models* to reflect the fact that several instances of the same activity can occur in different business processes. The second assumption is that the whole system functionality must be divided into modular components so that the executable models, which invoke several of these components, can be freely configured; these modular components are the so-called functional operations. The approach is shown in figure 6.2: from partial activity models and a library of functional operations, the system can

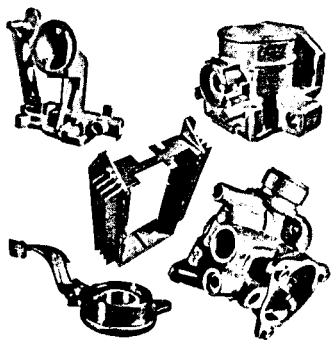


Figure 6.1: Sample parts manufactured by SONAFI[102]

[102]Courtesy of SONAFI

be configured by building a process model in which each activity invokes a certain functional operation. The workflow enactment service and the integration infrastructure are the run-time components that allow such kind of processes to be executed.

This approach bears some relationship to the GERAM framework shown earlier in figure 4.6 on page 147: the partial activity models are a kind of Partial Enterprise Model (component 5), while the functional operations can be regarded as being Enterprise Modules (component 8). In the PRONEGI project, however, since each activity invokes a functional operation then each functional operation can be defined as the system functionality required to support a certain kind of activities. Therefore, once the partial activity models were identified from the study of TQM processes, these activity models were subsequently refined in order to specify the set of required functional operations.

6.1.1 The functional operations

One of the fundamental processes in quality management is the control of *nonconformities* [ISO, 1999]. Basically, a nonconformity is a discrepancy between some product feature and the original specification for that feature; for example, the diameter of a certain part may fall outside the acceptable range which, in turn, has been specified from customer requirements. Nonconformities are thus central to quality management since, if not handled properly, they will have a direct impact on customer satisfaction. Whenever a material or product shows a nonconformity, there are usually only three possible courses of action: (1) the product is recovered by repair or rework, (2) product use, within specific limits such as time and quantity, is consented by a written authorization called a *deviation permit*, and (3) the product is rejected and so it may be recycled, disposed of, or returned to its original manufacturer.

SONAFI has an extensive Quality Procedures Manual, which specifies how quality management issues must be handled within the enterprise. In particular, there is a specific process for nonconformity treatment, which is usually abbreviated to TNC[104]. The TNC process, which is shown in figure 6.3(a), specifies that, once a nonconformity is found, the product must be immediately labeled as suspended and the nonconformity must be recorded and sent to the Quality Department. The Quality Department will create a team by calling members from several departments, if necessary, and this team will start by identifying possible causes for the nonconformity. Afterwards, the team will de-

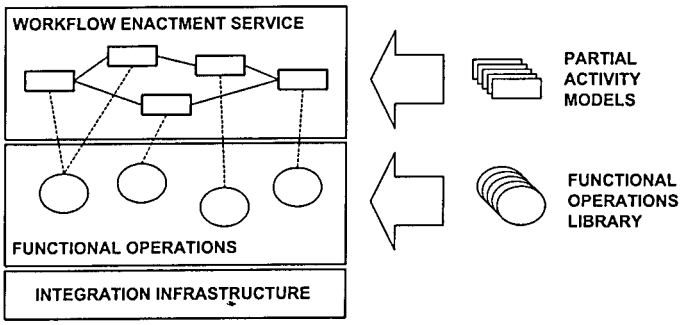


Figure 6.2: The PRONEGI system approach[103]

[103] [Macedo and Ferreira, 1998]

[104] This acronym, as well as others in this section, a due to the Portuguese translation of these terms.

cide how the nonconformity should be handled: whether the product should be rejected, recovered, or used according to a deviation permit. In this last case, there is an administrative procedure for requesting and approving a deviation permit. Once the treatment is decided, the Quality Department will validate the decision and request the Production Department to perform that treatment.

Another important process described in the Quality Procedures Manual is customer claim/return treatment (TR/D), which is shown in figure 6.3(b). Whenever a customer finds one or more nonconformities in a delivered product, the customer submits a claim to SONAFI. The claim may be received by several entities within the enterprise, but any such entity must forward the claim to the Commercial Department. Once the Commercial Department receives the claim, it creates a document called Claim/Return Bulletin (often abbreviated to BR/D). If the customer returns the supposedly defective products, these products are received by the warehouse and the return is recorded in the same BR/D bulletin. The Commercial Department sends the BR/D bulletin both to the Quality Department and to the Production Department. The Production Department will be subsequently involved in analyzing the claim and performing any corrective actions deemed necessary. The Quality Department will gather together a team to analyze the claim and the return products if available. The process goes on much like an internal TNC process but, in this case, the customer will be informed of the decided treatment.

6.1.1.1 Activity models and functional operations

These processes have been modeled according to the ARIS methodology [D. Ferreira, 2000]. The ARIS methodology [Scheer, 1992] comprises four different views: the organizational view, the data view, the control view, and the function view. Figure 6.3 shows simplified control views for the TNC and TR/D processes, respectively, using the event-driven process chain notation. From these diagrams it is possible to identify common activities such as "create team", "approve treatment", or even "analyze nonconformity", which is equivalent to "analyze claim" in the TR/D process. Apart from the details of each process, these common activities are basic units of work that may be present in several processes: they can be regarded as mechanisms with a definite purpose, that produce some output given some type of input. This kind of model represents the partial activity models that PRONEGI was looking for. Rather than considering the sum of all activities from different processes, PRONEGI aimed at identifying a small, minimal set of partial activity models that can be used to build those processes. These partial activity models have been expressed as a set of UML Use Case Diagrams [Pereira, 2000].

From these activities it was possible to identify the required functional operations. Basically, a functional operation supports one or more human users in performing their tasks. In general, the functional operation should implement a portion of system functionality that is both necessary and sufficient for a given task. At the same time, it should be possible to use one functional operation without requiring the use of other functional operations. Therefore, the set of functional operations has been developed as a set of modular software components that implement essential capabilities. These capabilities assist the user in tasks such as, but not limited to, creating customer claim records, creating BR/D bulletins, choosing multi-disciplinary teams, registering possible causes for a nonconformity, and deciding how nonconformities are to be handled.

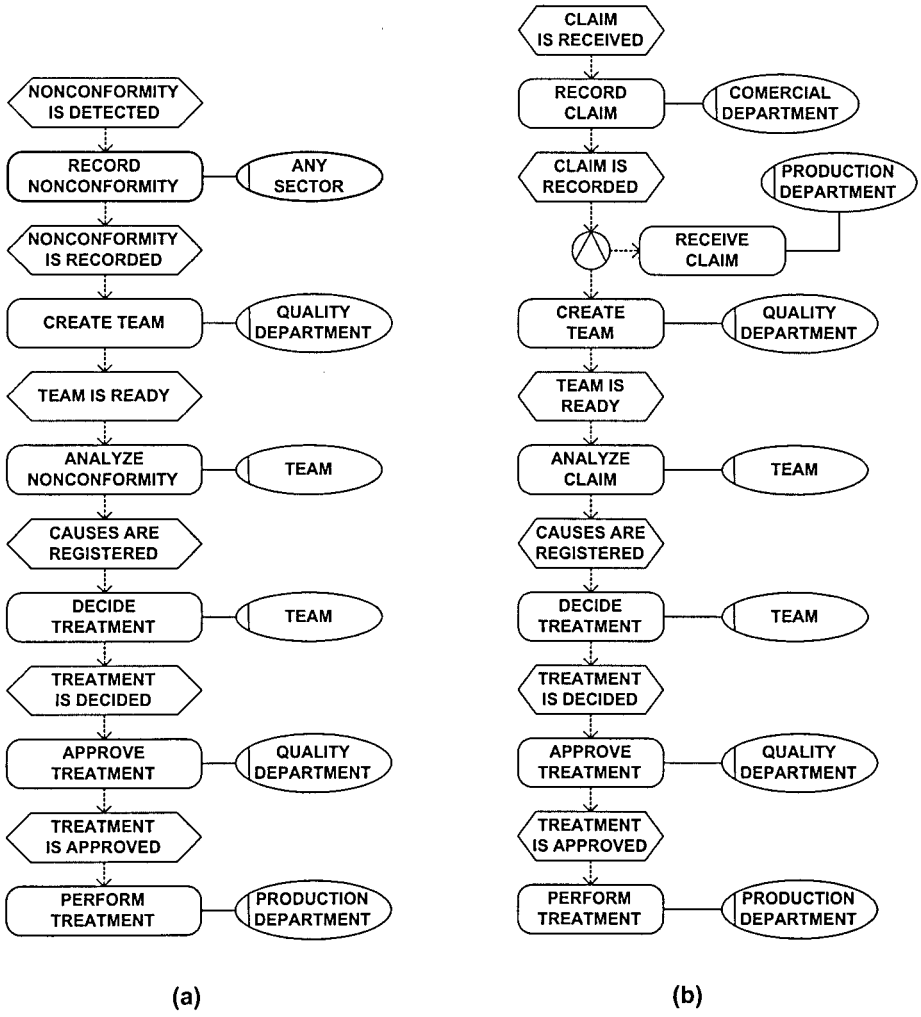


Figure 6.3: Simplified TNC (a) and TR/D (b) processes using ARIS eEPC notation

6.1.1.2 3-Tier client/server architectures

By the time PRONEGI was being developed, in early 2000, there was a considerable upheaval in the way enterprise information systems were being designed and implemented. As discussed previously in section 2.3.3.2 on page 55, it became evident that the Internet and its associated technologies would have an impact on enterprise information systems as well. At first, it seemed natural that the information system functionality should be accessible via the Web browser rather than via special-purpose client applications, as it was (and still is) usually done. At the same time, however, this was just one of the consequences of a different kind of architecture for distributed applications. This new architecture was a client-server architecture as before, but now with three layers instead of two: the user interface layer, the business logic layer, and the data layer. This is known as the three-tier client-server architecture. The main contenders in this area, who were trying to push their own technologies, were Microsoft with Windows Distributed interNet Applications (Windows DNA) and

Sun Microsystems with Java 2 Enterprise Edition (J2EE) [Thomas, 1999]. From these two, only J2EE remained within its original form.

These circumstances have had a strong influence on the development of PRONEGI's functional operations. Whereas originally these were thought to be CORBA components, now it seemed more appropriate to employ an architecture such as J2EE, which provided an extensive set of APIs and far better development and deployment tools, besides being a state-of-the-art technology for distributed applications. Following this trend, the functional operations have been implemented as a set of Enterprise JavaBeans (EJBs) with JavaServer Pages (JSP) front-ends. The EJBs access a database, which maintains all quality management data, via Java Database Connectivity (JDBC). Incidentally, this database is exactly the same database for the Quality Toolkit, the previous client-server system that INESC Porto had developed for SONAFI. Thus, the PRONEGI system manipulates exactly the same underlying data but, being founded on workflow management and functional operations, provides a degree of flexibility that was previously unavailable. The EJB components have been deployed using the Orion Application Server[105].

6.1.1.3 User interface

The user interface for one of the functional operations is shown in figure 6.4. Regardless of their particular user interface elements, every functional operation has two buttons: the "Save" button is pressed when the user completes the activity it was supposed to do, while the "Cancel" button is used if the user cannot complete the activity due to some reason, such as system failure. In general, a

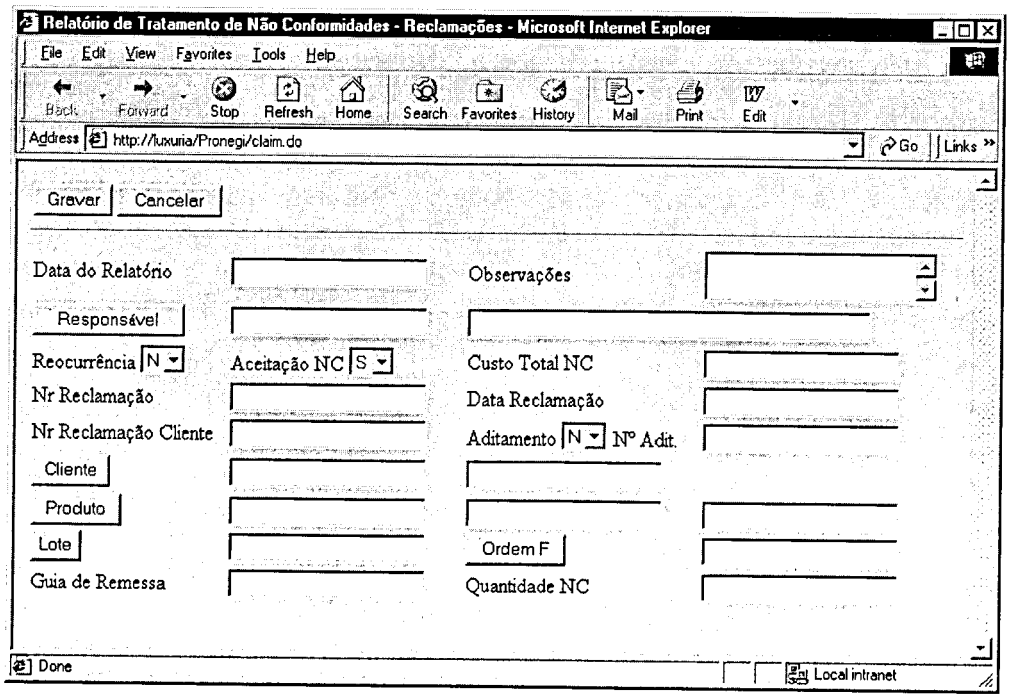


Figure 6.4: User interface for a functional operation (Customer Claim)

[105] http://www.orionserver.com/

functional operation gathers the data supplied by the user and, when the user presses "Save", it inserts, updates or removes that data from the quality management database. If, after several attempts, the database operation cannot be successfully performed, the user can press the "Cancel" button, denoting the fact that it was unable to fulfill its task. Both buttons make the functional operation generate a message denoting one of these events.

6.1.2 The Workflow Backbone

As shown in figure 6.2 on page 281, the PRONEGI system has three different components: the functional operations, the integration infrastructure, and the workflow enactment service. Chronologically, however, the integration infrastructure was the first to be designed, and a prototype implementation was already being undertaken by the time J2EE was chosen as the technological architecture for the functional operations. The project proposal had been prepared before architectures such as J2EE had emerged, and the PRONEGI system was expected to be built as a distributed application platform based on CORBA. But once J2EE appeared, it just seemed much more convenient than any other alternative as far as functional operations were concerned. Since the integration infrastructure was already being implemented with CORBA, it was necessary to reconcile the development of functional operations with that of the integration infrastructure.

This integration infrastructure is called the Workflow Backbone (WfBB) [H. Ferreira, 1999]. The WfBB is intended to provide the underlying infrastructure for a distributed workflow management system. This workflow management system can be described as comprising three layers: (1) the process modeling and execution services, (2) the infrastructure supporting process execution, and (3) additional supporting services. These layers correspond to the EMEIS layers presented earlier in section 4.3 on page 150. The WfBB is the infrastructure supporting process execution, and its purpose is to provide a communication channel that supports the interaction between the different elements within a workflow management system.

In view of the workflow reference model depicted in figure 3.3 on page 67, the WfBB can be regarded as the infrastructure that connects the workflow enactment service to the remaining components. However, this conceptual framework has been somewhat reshaped in the PRONEGI project. First, the WfBB provides a single interface rather than several interfaces between different components. Second, the process definition tool and the administration and monitoring tools have been built into the workflow enactment service, so interfaces 1 and 5 are implicitly implemented within that application. Third, as will be discussed ahead, functional operations are invoked from workflow clients and they return result data to the WfBB. Fourth, it is possible for other workflow enactment services to connect to the same infrastructure, although this has not been investigated within PRONEGI. The resulting workflow architecture is shown in figure 6.5.

6.1.2.1 Publish-subscribe communication

The WfBB is an object-oriented infrastructure that allows different kinds of applications to exchange messages. In this respect, the WfBB can be regarded as being a MOM system. This MOM system includes basic software components that have been purposely designed to support workflow execution, hence the name of Workflow Backbone. A fundamental feature of the WfBB is that it does not actually move information between applications, it only delivers references to objects that contain the desired information. Furthermore, its operation is based on allowing components to register their

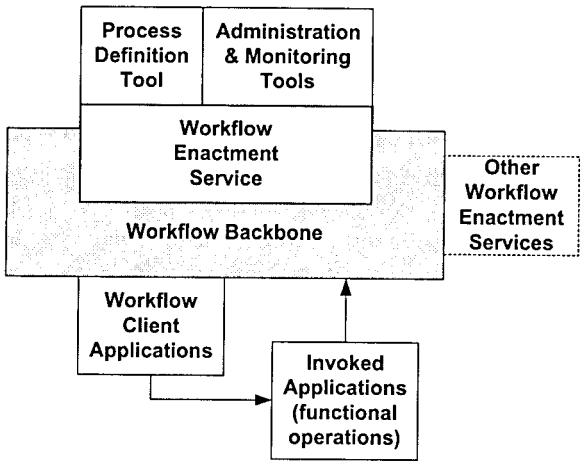


Figure 6.5: The PRONEGI workflow architecture

interest in receiving information from other components. Thus, according to figure 5.2 on page 169, the WfBB is a publish-subscribe system.

The *observer pattern* [Gamma et al., 1995] was the basis for the design of the WfBB. This design pattern defines two main participants, as depicted in figure 6.6: a Subject and an Observer. The Subject has an arbitrary number of Observers, which the Subject notifies whenever there is a relevant change or event. Each of the Observers can be attached and detached from the Subject at run-time, so that they can start or stop receiving notifications from the Subject.

The observer pattern is usually employed when there is a dependence between objects so that a change in one object requires changing other objects, or when one object should be able to notify other objects without making assumptions about these objects. Both of these situations occur simultaneously in a workflow management system. For example, the former occurs when a resource completes an activity and the workflow enactment service must be notified of that event, and the latter occurs when the workflow enactment service notifies the resources assigned to the following activities. These and other behavioral requirements, such as the fact that a process may be triggered by some event, have been the motivation for adopting the observer pattern.

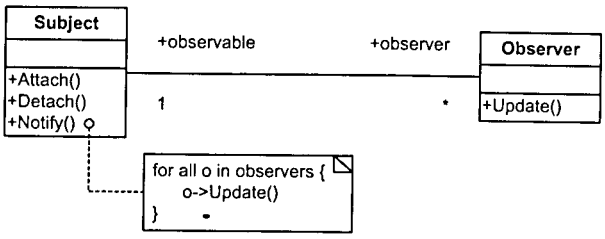


Figure 6.6: The observer pattern[106]

[106][Gamma et al., 1995]

The WfBB has enhanced the observer pattern in order to support publish-subscribe communication between a workflow enactment service and the resources assigned to individual activities. From this point of view, a Subject represents an activity that is sent from the workflow enactment service to its assigned resource. The Subject may also carry result data once the resource completes the activity. Thus, a Subject represents any kind of data that is exchanged between the workflow enactment service and the workflow participants. The exchange of a Subject takes place between a Publisher and a Subscriber. The Publisher is the one who sends the Subject, and the Subscriber is the one who receives it. A Subscriber is just a special kind of Observer. The Publisher is the one who produces the change or event (i.e. the Subject) that is notified to the Subscriber.

Every workflow participant plays the role of Publisher and Subscriber at some point during process execution. The workflow participant is a Subscriber when it receives an activity, and a Publisher when it returns the result. The workflow enactment service is a Publisher when it dispatches an activity, and a Subscriber when it receives the result. The activity and the result are just different Subject instances. In general, a client application (referred to also as a WfBB user) may have any number of Publishers and Subscribers, but a single Publisher and Subscriber will be enough for most users.

6.1.2.2 The use of Contexts

A distinctive feature of the WfBB is that it assumes that each activity is performed under a certain Context. Different activities may be performed under the same Context or under different Contexts. The Context defines which workflow participants will be able to receive and perform the activity. An activity that is sent under one Context will be received by the Subscribers within that Context only. If all Subscribers perform under the same Context, then all of them will receive all activities, and this corresponds to the original observer pattern, where all Observers are able to receive the Subject. On the other hand, if there is one Context for each activity, this corresponds to the fact that each activity is assigned to a single, separate workflow participant, and only that participant will be able to receive and perform the activity. Between these two extremes, a Context can be used to identify a group of workflow participants with similar responsibilities or capabilities.

The Context applies to every Subject in general, since an activity is just a special kind of Subject. Therefore, a Subject is always exchanged between a Publisher and one or more Subscribers under a certain Context, as suggested in figure 6.7. This Context is specified when the Publisher or Subscriber is created. From this point onwards, the Publisher or Subscriber will be able to send or receive Subjects within that Context only. The Context is also the basis for security enforcement within the WfBB: the access to Contexts is controlled by means of username and password. Thus, when a workflow participant logs into the WfBB, it must provide these elements before it can send or receive any Subject. The Context under which workflow participants exchange Subjects is usually defined at an administrative level, and cannot be changed by the users themselves. An exception to this rule is granted to the workflow enactment service, which must be able to control the execution of activities across all Contexts.

It is possible for a Context to contain other Contexts. This relationship between Contexts is hierarchical and it allows Contexts to be grouped together. The Contexts at the bottom of the hierarchy can be regarded as local, whereas those at the top are global. For example, Context X may represent a group of workers which can be further subdivided into sections 1, 2, and 3. If Context X is defined as containing Contexts 1, 2, and 3, then any Subject sent under Context X is received by Subscribers within Contexts X, 1, 2, and 3. But if a Subject is sent under Context 1, then only the Subscribers of that Context will receive it. This structure is called the Context tree, and Contexts in this tree are

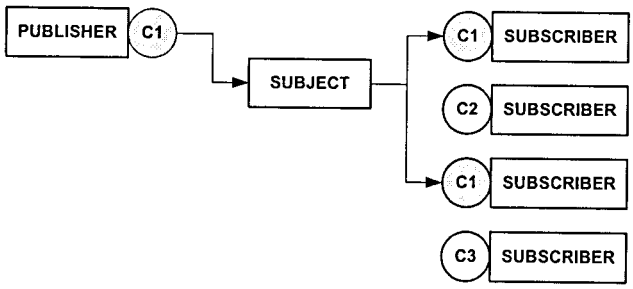


Figure 6.7: Publish-subscribe communication using Contexts

often referred to as child Contexts and parent Contexts. In essence, any Subject sent within a given Context is received by the Subscribers of that Context and by the Subscribers of any child Context.

Every time a Subject is published under a given Context, the WfBB will have to determine which Subscribers it should deliver the Subject to. Each Subscriber may perform under a different Context but, due to the relationships between Contexts, a Subject may have to be delivered to more Subscribers than the ones who perform under that Subject's Context. Whenever a new Subject is to be delivered, the WfBB uses the two-step procedure illustrated in figure 6.8. The first step is to build a list of admissible Contexts for each and every Subscriber: the first element in the list is the Context that the Subscriber performs under, and the following elements are found by climbing the Context tree. The second step is to deliver the Subject only to those Subscribers who have the Subject's Context in their list. Because the WfBB only retrieves one parent Context as it travels upwards through the Context tree, it is not allowed for a Context to have more than one parent Contexts, as suggested in figure 6.8.

6.1.2.3 Support for functional operations

If the WfBB is regarded as a MOM system, then the Subject represents the message exchanged between a Publisher and one or more Subscribers. This Subject carries workflow relevant data, such as a request to perform an activity. In this particular situation, and according to the PRONEGI architecture, the Subject should carry some information about the functional operations required to support the requested activity. In practice, as a result of the business process analysis described in section 6.1.1.1, it was possible to develop a single functional operation for each activity, so there is a one-to-one relationship between workflow activities and the functional operations that support them. As

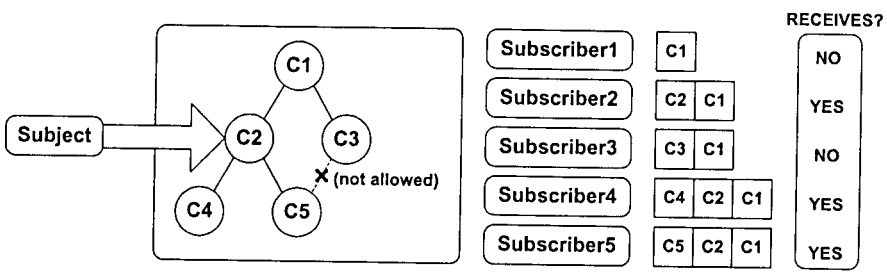


Figure 6.8: Procedure for determining whom to dispatch a Subject

a result, each Subject carries information about a single functional operation, which is the functional operation required to perform the activity being requested. However, there is no restriction in the WfBB architecture that could prevent the use of multiple functional operations for each activity.

The WfBB has one object class called FuncOper, which encapsulates information about functional operations. This kind of object is carried as an attachment to a Subject, and it can be extracted after the Subject is delivered to the Subscriber. A Subject may or may not carry an attached FuncOper object, depending on whether the Subject represents an activity request or not. At the time the WfBB was being designed it was still uncertain how the functional operations would be implemented, so FuncOper was specified as being able to support several kinds of software applications, including Java applications, Web-based applications, and native code. If necessary, FuncOper allows the Subscriber to download executable code in order to run locally the required functional operation. For example, it is possible to retrieve a Java class definition and to create a local instance of that class, or to transfer a specific Java object running on another machine to the local machine. Therefore, the WfBB supports *code mobility*, which is one of the challenging requirements for mobile agents, as discussed in section 5.3.1.1 on page 214.

In order to support these features, the WfBB introduces a separate component called the Functional Operations Manager (FuncOpsManager). Given the details of the functional operations available on the system, the FuncOpsManager is able to create FuncOper objects, which contain the information required to access the executable code for those functional operations, as illustrated in figure 6.9. Before dispatching a Subject, the Publisher requests the FuncOpsManager to create a FuncOper object. Then, the publisher attaches the FuncOper object to the Subject and dispatches the Subject to the Subscribers. When a Subscriber receives the Subject, it extracts the FuncOper object and requests it to retrieve the functional operation's executable code. The Subscriber can then run the functional operation locally.

The FuncOpsManager is as far as PRONEGI gets to the functional operations library shown earlier in figure 6.2 on page 281. The WfBB allows any number of FuncOpsManagers to coexist, so that functional operations can be arranged according to their purpose. There is one FuncOpsManager for functional operations supporting quality management processes but, should it become appropriate or necessary, new functional operations could be devised to support maintenance processes, for example. These new functional operations would be available from a different FuncOpsManager.

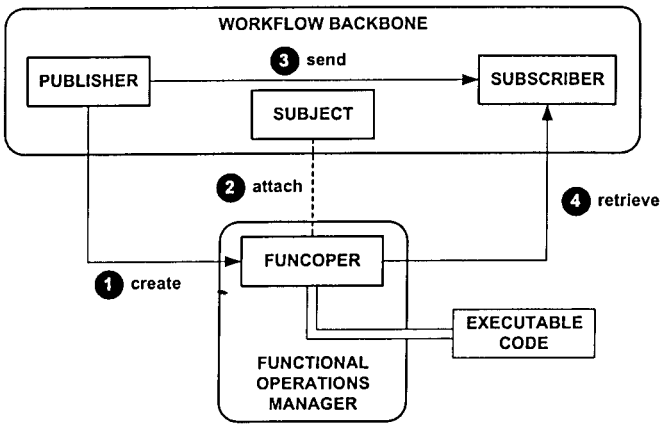


Figure 6.9: The FuncOpsManager and the Workflow Backbone

Figure 6.10 shows the user interface for the FuncOpsManager developed within PRONEGI. On the left hand side, the FuncOpsManager lists the available functional operations as well as the FuncOper objects that have been created. In this pane, the "objects" node represents the FuncOper instances that have been created, while the "connected objects" node list the instances that are available for immediate use. The purpose of this distinction was to create FuncOper objects that could maintain their state even if deactivated, but this possibility has not been exploited further. On the right hand side there are two panes for displaying the user interface of functional operations: one is able to display the graphical elements of a Java application, the other is able to display an HTML page. On the bottom, a text pane tracks the actions of this FuncOpsManager for debugging purposes.

Anticipating a possible Web-based implementation for functional operations, the FuncOper object class was defined with two additional methods that allow a Subscriber to access a Web front-end. One method retrieves a whole HTML page in order to display it locally; the other method retrieves just the URL. The first method was intended to be used by client applications with some capabilities of displaying Web pages. The second method assumes that the Web browser will be used instead. The implementation of these methods is similar to the previous ones, the main difference being that text is being transferred instead of byte code or executable code. It is worth noting that, when requesting the FuncOper object to retrieve an HTML page, the method call is effectively replacing the function of the HTTP protocol.

Eventually, the FuncOper method that retrieves a URL would be the only one to be used within PRONEGI, since functional operations have been implemented according to the J2EE architecture and they all have a Web-based user interface. The option of developing special-purpose client applications that would be able to display Web pages was considered at first, but it was discarded once it was realized that sophisticated user interface features were required, and that the Web browser would be more appropriate for that purpose.

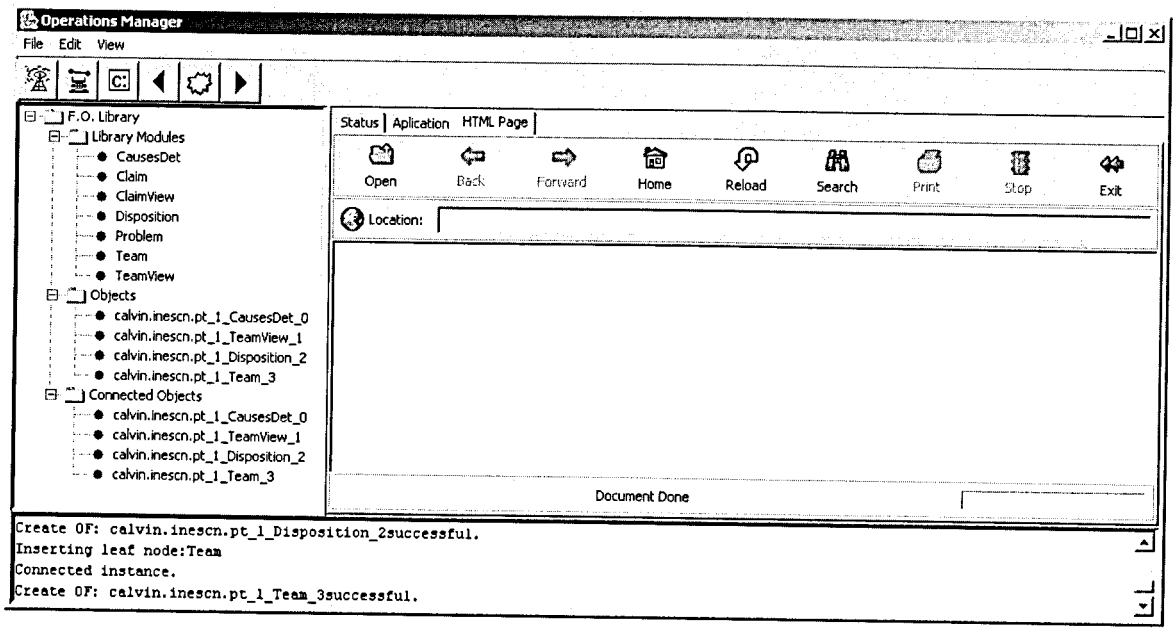


Figure 6.10: The Functional Operations Manager user interface

6.1.2.4 System design

The WfBB has been designed as an object-oriented system containing four packages - Repositorium, FunctionalOperation, ControlSystem, and SystemUtilities - which divide its functionality into four functional groups:

- the Repositorium package contains one class, called Repository, which is both the entry point and the main interface for client applications: this class allows client applications to perform basic actions such as creating Publishers and Subscribers, retrieving references to existing objects, and configuring user information;
- the FunctionalOperation package contains the FuncOpsManager and FuncOper classes, which are domain-specific, i.e., they have been included to represent particular entities of the PRONEGI architecture;
- the ControlSystem package contains the main classes of the WfBB, including Publisher, Subscriber, Subject and Context, that allow users to communicate with each other according to a publish-subscribe paradigm;
- the SystemUtilities package contains only abstract base classes which isolate common features from other classes, and represent the fundamental structure of the observer pattern; internally, in some situations, the WfBB uses references to these interfaces rather than to concrete objects.

These four packages and their classes and presented in figure 6.11.

First and foremost, the observer pattern is implemented by means of the Observer and the Observable classes. The Observer class has a single method called receive(), which receives a reference to an Observable object. This is the basis for Subject delivery within the WfBB: the WfBB requires all Subscribers to implement the Observer interface, and it will invoke the receive() method whenever a new Subject is available for delivery. Thus, Subjects are delivered via *callback*. The receive() method has the form:

```
void receive(Messenger, Observer, Observable)
```

where the first input parameter contains a reference to the Publisher who sent the Subject, the second parameter is a reference to a Subscriber to whom replies should be addressed, and the third parameter is a reference to the Subject. Upon reception, all these references must be narrowed to their actual object types.

The receive() method is invoked by a singleton Dispatcher object, which is the messaging engine of the WfBB. It is the Dispatcher object who receives Subject references from Publishers and dispatches them to Subscribers.

The Dispatcher object works in connection with another singleton object, the Repository, which keeps track of all objects within the WfBB. The main purpose of the Repository class is to implement the methods for creating and retrieving Publishers, Subscribers, Contexts and Subjects. When invoking these methods, the objects are always referred to by their a name. As a matter of fact, all objects within the WfBB have a name, except for the Repository object. This explains why the Identifiable class is at the heart of the WfBB: it defines the name as a common feature for all object classes.

Additionally, the Repository class allows client applications to retrieve references to the available FuncOpsManagers, in order to subsequently determine the functional operations available on the

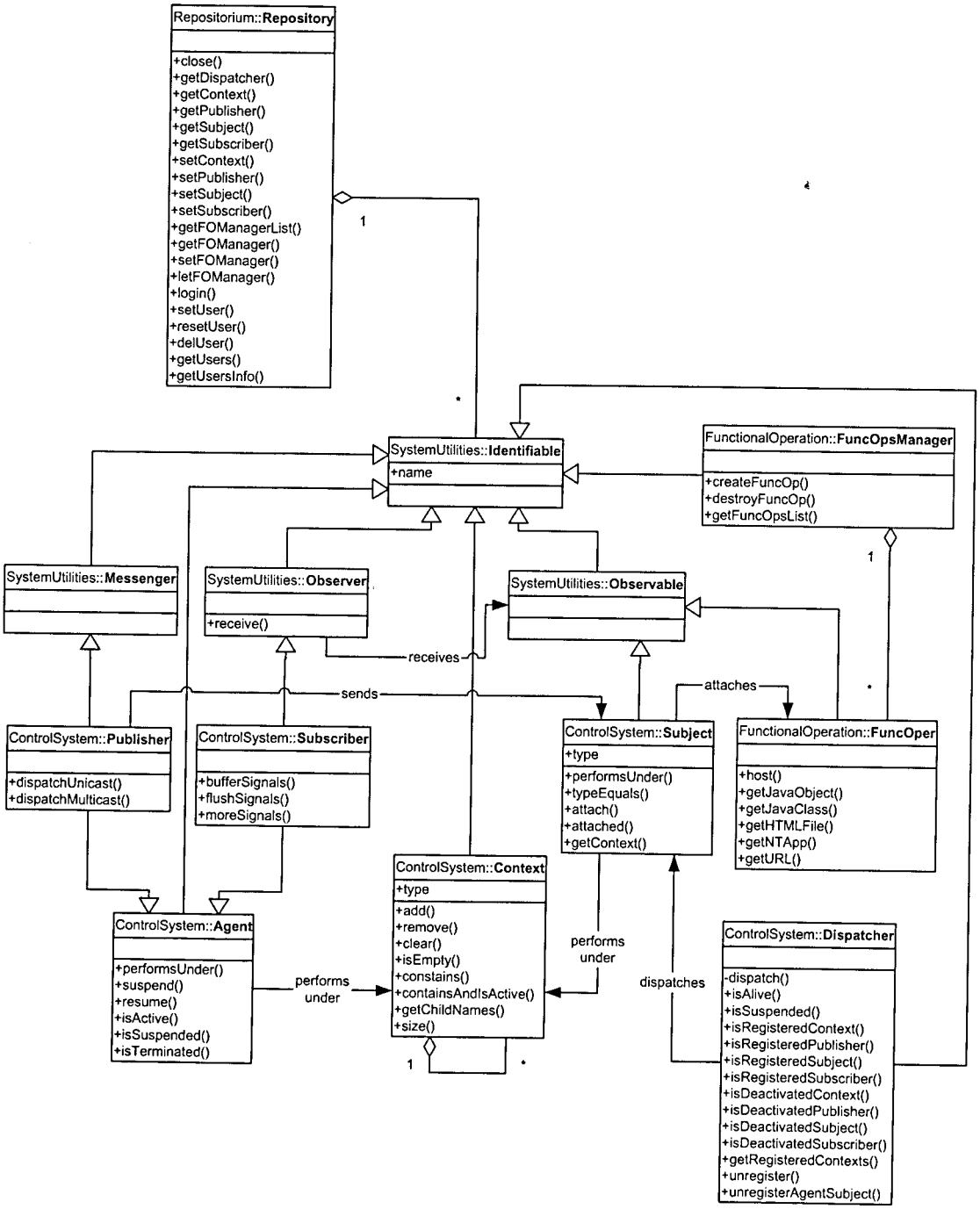


Figure 6.11: Class diagram for the PRONEGI Workflow Backbone

system. Internally, the Repository class maintains a list of FuncOpsManagers connected to the WfBB; these FuncOpsManagers are distinguished by name.

The Repository class also implements the methods that allow user information to be configured but, in general, only administrative tools will make use of these methods. Users will connect to the WfBB by providing their username and password when invoking the login() method.

The close() method in the Repository class shuts down the WfBB. The second method, getDispatcher(), allows client applications to retrieve a reference to the singleton Dispatcher object. On its turn, the Dispatcher object allows a user to retrieve a complete list of all registered Contexts. The most common use of this class, however, is when a client application wants to disconnect from the WfBB; in this case, the application will invoke unregister() in order to disable its Publishers and Subscribers. The Dispatcher will then set the state of these objects to "deactivated" and will prevent other users from exchanging Subjects with this application. If there are any pending messages for delivery, however, they will still be delivered before the Subscriber is definitely disabled.

The Subscriber is a subclass of the Observer class, so it receives a Subject as an Observable object. However, the Subject brings another Observable object attached, the FuncOper. The FuncOper objects do not live within the WfBB; instead they belong to a FuncOpsManager, which may be located in another machine. The host() method in the FuncOper class allows this machine to be identified. The remaining methods are intended to locate the executable code for the functional operation, as discussed in section 6.1.2.3.

On the other hand, the Publisher is a subclass of the Messenger class which, despite having an empty definition, has been included in order to play the role of Publisher within the SystemUtilities package. With some auxiliary message dispatching methods defined in the Messenger class, it was possible to test the messaging engine within the SystemUtilities package, and only afterwards to consider the remaining packages. The Messenger class was included also as a placeholder for future functionality.

In spite of their complementary functions, the Publisher and Subscriber classes have some features in common. Both of them work under a certain Context, and both can be temporarily deactivated by calling suspend(). The Dispatcher will not allow a deactivated Publisher to dispatch Subjects, or a deactivated Subscriber to receive Subjects, but they can be reactivated by calling resume(). This behavior has been implemented in a class called Agent, which is an abstraction of the features that are common to Publishers and Subscribers.

Besides the Agent class, another class that is associated with a Context is the Subject, which implements a similar performsUnder() method. The Subject also shares a common feature with the Context: the type attribute. Originally, it was thought that the type attribute would be useful in order to semantically arrange Subjects and Contexts according to their purpose. In practice, however, this attribute is used as an extra field to convey additional information within a Subject. In the Context class, the type attribute remained unused. The Context class methods, on the other hand, allow relationships between Contexts to be defined: the add() method adds a child Context, while the clear() method removes the relationship with all child Contexts (without deactivating those Contexts). There are also methods to test whether a Context is the parent Context of another, and to retrieve the names and number of child Contexts.

6.1.2.5 Dynamic behavior

Figure 6.12 illustrates how a workflow activity is dispatched, as a Subject, through the Workflow Backbone. In this example, the workflow enactment service is initiating the execution of a workflow

process with four activities; the workflow enactment service is about to request the execution of the first activity. For this purpose, the workflow enactment service must have previously requested the FuncOpsManager to create a FuncOper object that represents the functional operation that supports this activity. Figure 6.12 then shows that the workflow enactment service creates a new Subject (step 1) and attaches that FuncOper object to this Subject (step 2).

Afterwards, the workflow enactment service hands the Subject to the Dispatcher (step 3) by calling its Publisher's dispatchUnicast() or dispatchMulticast() method, depending on whether the activity is assigned to a specific Subscriber or whether it should be sent to all Subscribers within the Publisher's Context, respectively. In both cases, the Dispatcher collects the Subject reference and places it in the Process Queue. All Subjects remain temporarily in the Process Queue before the Dispatcher effectively determines which Subscribers the Subject should be sent to. If the Subject is to be dispatched in multicast mode, then the Dispatcher will use the procedure shown in figure 6.8. If the Subject is to be dispatched to a specific Subscriber (unicast mode), then the Dispatcher just checks that the Subscriber is registered and that it has not been deactivated.

As a result of this procedure, the Dispatcher will place in the Dispatch queue (step 4) one Subject reference for each Subscriber that should receive the Subject. In unicast mode the Dispatcher will place, for each Subject reference in the Process Queue, at most one Subject reference in the Dispatch Queue. In multicast mode the Dispatch Queue is loaded with several references of the same Subject, as many references as there are Subscribers that should receive it.

Apparently, the Subject could now be delivered to those Subscribers by calling their receive() method. However, this approach would suffer from two deficiencies. The first is that the Dispatcher would not be able to do anything else while the Subjects in the Dispatch Queue are being delivered. The second problem is that, because Subjects would be delivered one at a time, each Subscriber would have to wait for other Subscribers to receive their Subjects before it would receive its own. The solution to these issues is to request a separate thread to deliver the Subject to each Subscriber.

From time to time, the Dispatcher will walk through the Dispatch Queue in order to assign each Subject reference to the appropriate delivery thread (step 5). There is one delivery thread for each Subscriber. This delivery thread is launched when the Subscriber is created. The Dispatcher will place Subject references for the same Subscriber in the same delivery thread, and Subject references for different Subscribers in different threads. This way, each Subscriber receives Subjects from a single thread (step 6).

From a received Subject reference, the workflow participant will extract the FuncOper reference in order to retrieve from this object (step 7) the URL for the functional operation supporting the workflow activity being requested.

When a workflow participant replies to the workflow enactment service, the behavior is basically the same but with the workflow participant playing the role of Publisher, and the workflow enactment service that of Subscriber. In general, however, Subjects that are dispatched from a workflow participant to the workflow enactment service do not carry a reference to a FuncOper object. Usually, they convey only status information about a previously requested activity. It is important to note that, when a workflow participant successfully runs a functional operation, or when it fails due to a system or data error, the functional operation will automatically report this event to the workflow enactment service, so that it can proceed with the following activities or request the workflow participant to perform the same activity again.

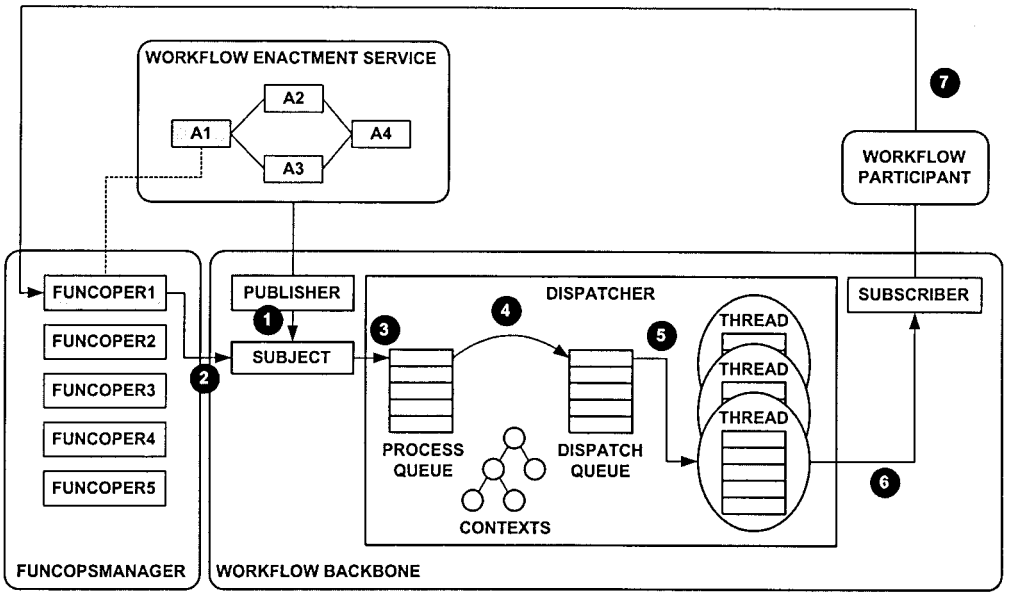


Figure 6.12: Dispatch sequence across the Workflow Backbone

6.1.2.6 System implementation

The purpose of the WfBB is to provide generic support for workflow process execution. It was devised as an infrastructure that integrates different software modules, which have been developed separately. As shown in figure 6.5 on page 286, these software modules are: the workflow enactment service, the workflow client applications, and the functional operations. During the development of the WfBB, several requirements were taken into account. First, it should be a scalable system, i.e., it should be able to handle an increasing number of Publishers, Subscribers and Subjects without a significant decrease in system performance. Second, it should be flexible, i.e., it must allow Publishers, Subscribers, Subjects and Contexts to be configured at run-time according to application requirements. Third, it should be open, i.e., it should provide the means for additional tools or applications to be easily connected to the same infrastructure.

For these reasons, the WfBB and its constituent components, shown in figure 6.11, have been implemented in Java. With Java technology (the language, the environment, and the development tools), the development of complex middleware systems is made easier than with alternative technologies. Besides, Java includes a complete set of APIs and function libraries, in fact much more than what is required as far as the WfBB is concerned, but nevertheless all that was required for its implementation. In practice, only basic language features and the CORBA API were necessary. The WfBB has a command-line user interface as shown in figure 6.13.

Given that the main goal of the WfBB is to promote interoperability between all connected applications, the choice of Java technology should not deprive the remaining applications of the possibility of employing alternative technologies. The WfBB should adopt a standard middleware architecture so that other applications, possibly using different technologies, can interoperate with the Java implementation. CORBA seemed the natural solution. Therefore, the Java classes shown in figure 6.11, which are the internal WfBB components, have been exported as a set of Interface Definition Lan-

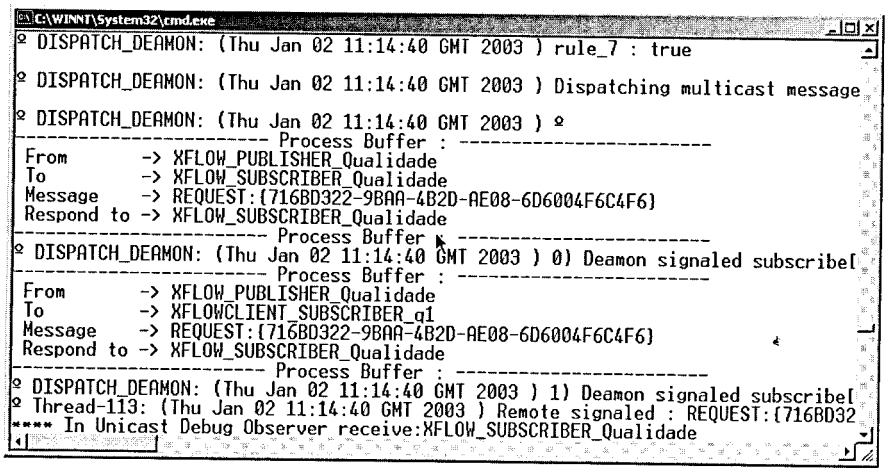


Figure 6.13: User interface for the Workflow Backbone

guage (IDL) interfaces. Thus, for each of those classes there is an IDL interface that provides access to the corresponding Java implementation. External applications, written in any language and running in any platform, can access and manipulate WfBB objects regardless of their implementation technology.

The question now was which ORB (Object Request Broker) implementation to use. Initially, development begun with the Voyager ORB which, at that time, was supplied by ObjectSpace[107]. This ORB was simple to use because it was able to automatically generate CORBA skeletons and stubs for Java classes. From the Java implementation it was possible, without further effort, to make the WfBB available via CORBA. Later, due to licensing issues, another ORB had to be chosen. At first, it was thought that the ORB supplied with the Java Development Kit (JDK) would suffice, so the WfBB implementation was adapted to this ORB. However, it was found that this ORB had occasional problems when communicating with other ORBs, so a third ORB had to be found.

Eventually the chosen ORB was JavaORB[108] which, besides being free for commercial use, seemed to operate perfectly with other ORBs. By the time the PRONEGI system was ready for demonstration, however, JavaORB was discontinued and became unsupported. A new version, which was a complete re-implementation of JavaORB, was being developed. This new version is called OpenORB[109], and it is currently the underlying ORB for the WfBB.

6.1.3 The workflow enactment service (XFlow)

In the PRONEGI project there is a single application that combines three different kinds of functionality: process modeling, process execution, and administration of the WfBB. This application is called XFlow [D. Ferreira, 2001]. The XFlow application has an intuitive, drag-and-drop graphical user interface that allows a human user to create processes, define their triggering events, assign individual activities, and configure user information. All these task can be performed within the same application environment. Whereas process modeling and execution capabilities are often combined within a

[107] http://www.objectspace.com/
[108] http://dog.team.free.fr/details_javaorb.html
[109] http://www.exolab.org/

workflow enactment service, the same does not usually happen with administrative tools. In XFlow, however, the process modeling and execution functionality depends strongly on WfBB objects, so it is more practical to administer these objects from within a single application.

6.1.3.1 Process modeling

The XFlow application allows any process from SONAFI's Quality Procedures Manual to be expressed as a sequence of workflow activities, provided that the functional operations that support these activities are available. One of the most important processes is the TR/D process, as discussed in section 6.1.1 on page 281. This process can be expressed as the sequence of workflow activities depicted in figure 6.14. Each workflow activity must be performed by a given department, and it is supported by a certain functional operation. Thus, in order to create a workflow activity, these two elements must be specified. The first step shown in figure 6.14 - "Customer Claim" - represents the event that triggers the whole process.

In order to request these activities via the WfBB, the XFlow application has to publish a Subject under a certain Context, the Subject carrying a reference to a FuncOper object that allows the workflow participant to retrieve the URL for the functional operation. The use of Contexts allows activities to be sent to one or more workflow participants, because any Subject that is published within a Context is dispatched to all Subscribers of that Context. This possibility turns out to be quite practical, since a Context can be used to represent a group of human resources with similar responsibilities. In PRONEGI, Contexts have been used to represent SONAFI's departments - the Commercial Department, the Quality Department, and the Production Department. Instead of assigning activities to individual users, the workflow activities are assigned to these Contexts.

This approach is advantageous because there is no need to specify an individual user, and the performer for each activity can be any of the Subscribers within a certain Context. Thus, the user is found dynamically at run-time rather than being pre-defined. What must be ensure is that each activity is dispatched to the right department, regardless of who in that department will perform the activity. This reflects current practice at SONAFI, where there is a good reason for this behavior: the fact that there are shifts for the same position, so that the same job is performed by different persons at different times. In most cases, however, what happens is that it is clear which group of persons should perform a given activity, but it is not certain who will be available from that group to effectively perform it at run-time. To some extent, this also depends on the workload assigned to each of those resources.

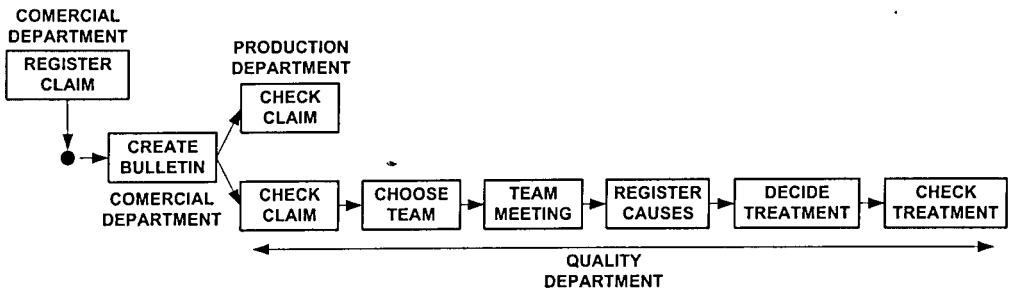


Figure 6.14: The TR/D process expressed as workflow process

Nevertheless, it is still possible to assign activities to individual users if each user subscribes to its own Context. In this scenario, it would be appropriate to configure the Context tree so that user groups are represented at the top, and individual user Contexts are defined as child Contexts. This approach still ensures some flexibility because the Context is fixed but, at different times, there may be a different Subscriber receiving Subjects on that Context. In conclusion, assigning activities to Contexts relieves the workflow designer from having to specify individual users, without restricting its ability to assign activities to particular users if necessary.

In the XFlow application, the process shown in figure 6.14 can be built in a drag-and-drop manner, in which a FuncOper object and a Context are combined in order to define a workflow activity. This functionality is illustrated in figure 6.15. On the upper-left corner, there is a window that displays all FuncOper objects available on the system. This information has been retrieved from a FuncOps-Manager connected to the WfBB. On the upper-right corner another window displays the registered Contexts, which correspond to three different departments. These two windows contain the elements that can be dragged and dropped in a Process Editor window.

In order to create a workflow activity, a FuncOper element must be dropped onto the Process Editor. The activity is assigned a unique identifier, in the form of a Globally Unique Identifier (GUID) [Chappell, 1996]. Subsequently, one of the available Contexts must be dragged and dropped on that activity. This means that the activity will be dispatched as a Subject under the specified Context. To establish the order of precedence between activities, these can be linked by means of drag-and-drop operations on the Process Editor window. Each activity contains the name of its functional operation below, whereas its color is the same as the Context it was assigned to. The icon shows the status of execution; in figure 6.15 no activity has been started yet.

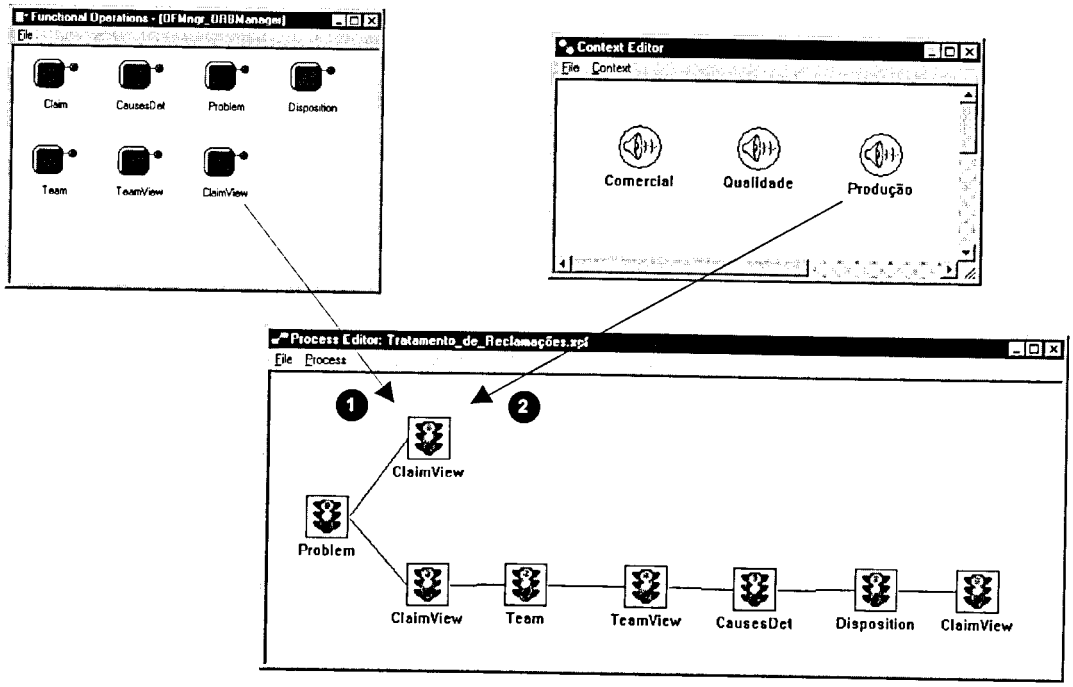


Figure 6.15: Defining the TR/D process in the XFlow application

6.1.3.2 Activity parameters

The FuncOper object and the Context are all that is necessary in order to dispatch a Subject, which requests the execution of an activity, to a workflow participant. However, when invoking a functional operation the workflow participant may have to supply some input data to the functional operation. For example, after the Commercial Department creates the BR/D bulletin, depicted as the second step in figure 6.14, the Production Department and the Quality Department must know the number of that bulletin in order to check the claim since, at any given time, there may be several BR/D bulletins in the quality management database. Thus, functional operations may require input information coming from previous workflow activities.

In order to support this possibility, each activity has input and output parameters. These parameters are specified in two text strings, as shown in figure 6.16. The input parameters may be defined when the process is being modeled. In this case, each parameter should be given in the form name=value where name and value are both alphanumeric. If there are several input parameters, then they should be separated by an ampersand, as suggested in figure 6.16. The output parameters are returned from the functional operation according to the same syntax, once the functional operation successfully performs its action. During process execution, the XFlow application appends the output data coming from one activity to the input data for the following ones. This way, the information generated by one functional operation flows throughout the whole process.

6.1.3.3 Process triggering

Any workflow process can be triggered in two ways. One possibility is to trigger the process manually by invoking the menu command shown in figure 6.17(a). The second possibility, which is used most often in practice, is to allow the unsolicited execution of a functional operation to trigger the process automatically. This means that, for example, when the Commercial Department receives a customer claim it may voluntarily invoke a functional operation to register that claim in the quality management database. This is the first step in figure 6.14, although it is not represented in figure 6.15. Once the functional operation completes this action, it will send a special-purpose Subject to the XFlow application, which will automatically start the TR/D process. This option can be configured as shown in figure 6.17(b).

Two parameters are required in order to configure how a process should be triggered automatically. The first parameter in 6.17(b) is the name of the functional operation being invoked. Whenever this functional operation successfully completes its action, it is expected to publish, on the WfBB, a Subject containing its name. This Subject is published under a special-purpose Context, appropriately called "XFlowTrigger". There is only one Subscriber for this Context, which belongs to the XFlow application. Whenever this Subscriber receives a Subject it extracts the name of the functional oper-

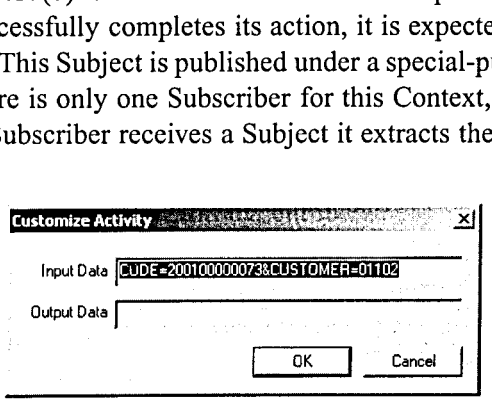


Figure 6.16: Activity input and output parameters

ation from the Subject, and it checks whether there is any trigger condition that has been specified with this name.

If there is such a trigger condition, then the XFlow application will read the second parameter in figure 6.17(b), which is the name of a file that contains the process to be triggered. XFlow will load this file and then create an exact copy, except for the fact that all activities will be set to inactive (i.e., not started yet). The process is started, and a new Process Editor window will open automatically so that the new process instance can be monitored.

6.1.3.4 User configuration

A similar facility is available for configuring the system users, as shown in figure 6.18. This graphical user interface is basically a front-end for invoking Repository class methods concerning user information. The user list displays the set of users registered in the WfBB, and when a new user is added, XFlow invokes setUser() in order to register that user in the WfBB. Three parameters are required for each user: a username, a password, and the Context the user will subscribe. In order to connect to the WfBB, the user must supply its username and password. The WfBB then authenticates the user and, if successful, returns a reference to the user's Context. The user will create a Publisher and a Subscriber for this Context and, from this point, it will be able to send and receive Subjects.

6.1.3.5 The workflow client

The PRONEGI system includes a special-purpose workflow client application which, besides connecting a user to the WfBB, also maintains a list of tasks for that workflow participant. This workflow

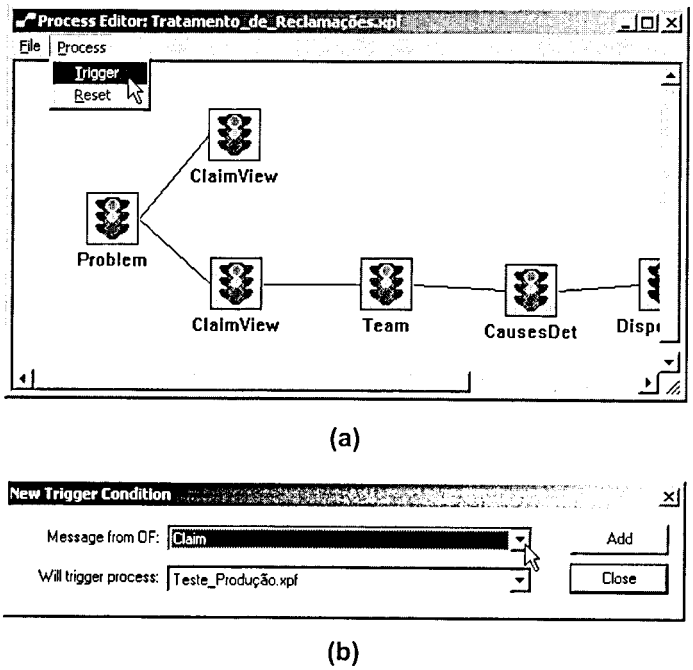


Figure 6.17: Triggering a process manually (a) and automatically (b)

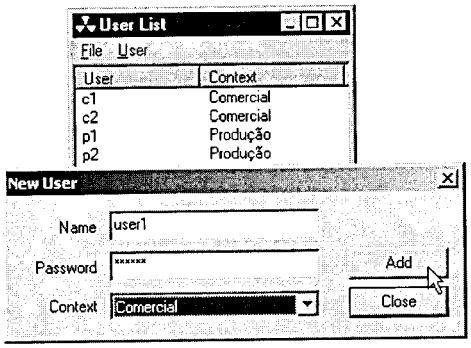


Figure 6.18: Configuring system users in the XFlow application

client application is sometimes referred to simply as the "client". It is similar in purpose, and even in appearance, to the Staffware Workqueue Manager (section 3.4.1.6 on page 93) or to the Floway client shown in figure 3.21 on page 102. In PRONEGI, there are two versions for the same client functionality, as shown in figure 6.19. When launching either one of these clients, the user is requested to supply its username and password (this step not shown in figure 6.19). If the user is successfully authenticated, the client automatically retrieves the user's Context and creates a Publisher and a Subscriber for that Context. Both clients display the Context name next to the title bar ("Commercial" in figure 6.19).

The workflow client maintains a list of all activities that are currently being requested or performed. These are kept as a list of Subjects. Each activity corresponds to a single entry in the task list, and its unique identifier is shown together with an icon that denotes its execution status. For each activity being requested, the user is able to accept it or reject it. When the user double-clicks on an accepted activity the client will extract, from the corresponding Subject, the attached FuncOper object. From this object, the client retrieves the URL for the functional operation and invokes the

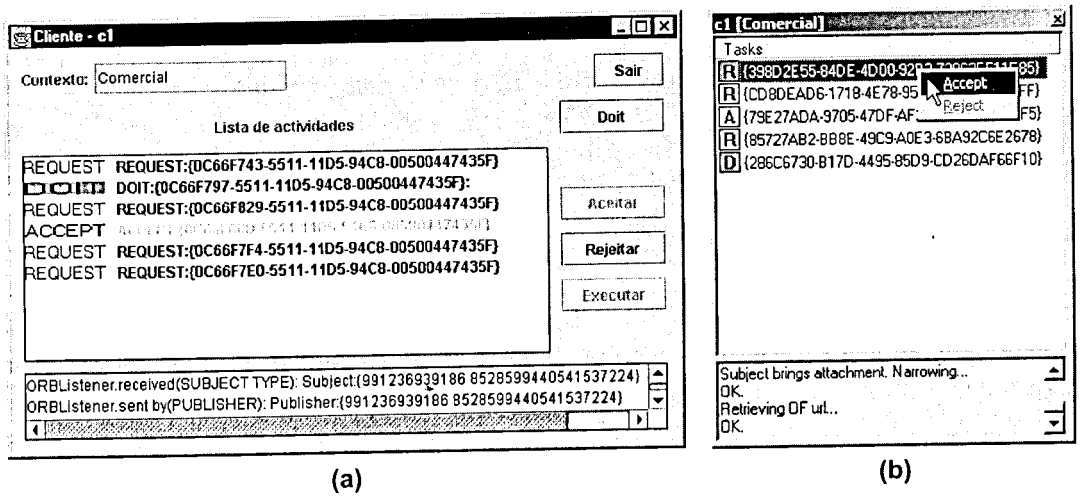


Figure 6.19: Workflow clients in the PRONEGI system

BIBLIOTECA F.E. PORTO

Web browser. The user is then presented with a user interface such as that shown in figure 6.4 on page 284.

It should be noted, however, that the client appends some additional information to the functional operation's URL before opening the URL in the Web browser. Basically, the client appends three elements to the URL: the activity's unique identifier, the user's Context, and the input data for the functional operation. With the input data shown in figure 6.16, the resulting URL would resemble the form:

```
http://host/Pronegi/functional.operation?activityID={10240686-C3EB-11D5-94F4-
00500447435F}&contextID=Commercial&CODE=200100000073&CUSTOMER=01102
```

The functional operation will read the parameters contained in this URL before sending back a response to the Web browser. The input data will be used to customize the user interface, or to retrieve some data from the quality management database prior to displaying the user interface. The activity identifier and the Context will be used when the user presses one of the buttons "Save" or "Cancel", as discussed in section 6.1.1.3. Once this happens, the functional operation creates a new Subject containing the activity identifier together with some status information, and it publishes this Subject under the specified Context. The XFlow application will receive this Subject and it will proceed to the next activities or request the same activity again, depending on whether it was successfully completed or not.

6.1.3.6 Activity assignment

Within the XFlow application, each activity is assigned to a Context and the activity is dispatched as a Subject to all Subscribers of that Context. However, only one of these Subscribers will actually perform the activity. The XFlow application and the workflow clients, which communicate via the WfBB, follow a simple assignment protocol that determines which workflow participant each activity will eventually be assigned to. This protocol is illustrated in figure 6.20. Whenever the XFlow application requests an activity, it publishes a Subject named "REQUEST" in multicast mode, under that activity's Context (step 1). The activity shows up in all workflow clients that subscribe to that Context. However, the Subject carries no attachment yet.

At this point, at least one user is expected to accept the activity by returning a Subject named "ACCEPT" in unicast mode to XFlow's Subscriber only. It should be noted that it is not possible that two users accept an activity at the same time because, as illustrated in figure 6.12 on page 295, there is a single thread delivering Subjects to each Subscriber, so the XFlow's Subscriber never receives two replies at the same time. The first user who accepts the activity (step 2) is the user that the activity will be assigned to.

To assign the activity to the user who accepted it, XFlow will dispatch two Subjects (step 3): one is named "DO IT" and it is dispatched in unicast mode to the user, the other Subject is called "CANCEL" and is dispatched in multicast mode to all Subscribers to the activity's Context. The first Subject confirms that the user has been assigned the activity, and it carries the FuncOper object containing the URL for the functional operation. The second Subject is intended to remove the recently requested activity from the task lists of the remaining users. The user that has received the "DO IT" Subject ignores the "CANCEL" Subject; all other users remove the activity from their task lists, regardless of whether or not they have meanwhile accepted the activity too.

The fact that an activity must be accepted before it is effectively assigned to an individual user allows users within the same Context to distribute their workload more evenly. For example, if a

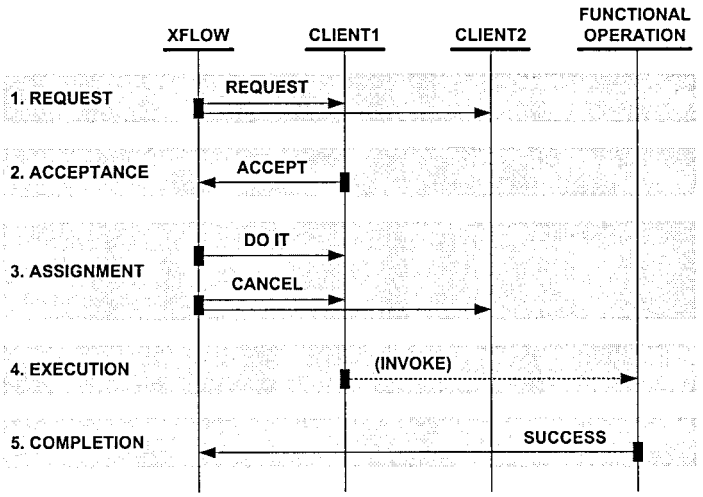


Figure 6.20: Activity assignment protocol in PRONEGI

user is busy, then another user is more likely to accept a requested activity. On the other hand, users could deceive this approach if they mutually agree to become less responsive to activity requests. Nevertheless, it is possible for a manager to monitor pending activities by subscribing to the same Context or to a child Context.

Despite having accepted an activity, a user is still free to reject it. As illustrated in figure 6.21(a) if the activity has been accepted but the user has subsequently rejected the activity, then the XFlow application will automatically go back to step 1 in figure 6.20 and it will request the same activity again to all Subscribers within the activity's Context. Figure 6.21(b) illustrates another kind of exceptional behavior. In general, functional operations will return a Subject named "SUCCESS" upon activity completion. However, if the functional operation returns a Subject named "FAILURE", which denotes an error during activity execution, the XFlow application will send another "DO IT" Subject to the same user, so that the activity is repeated. This is equivalent to going back to step 3 in figure 6.20 but without sending any "CANCEL" Subject. In this case, the user is again able to reject the activity.

The activity assignment protocol involves the XFlow application, the workflow clients, and the functional operations. These components exchange certain Subjects with a pre-defined name. In fact the Subject name has the form message:activity:data where message is one of "REQUEST", "ACCEPT", "DO IT", "REJECT", "SUCCESS", or "FAILURE", and activity represents the activity's unique identifier. If message is "DO IT", then the Subject carries a FuncOper object attached to it, and the data portion contains the input parameters for the corresponding functional operation. If message is "SUCCESS", the data portion contains the output parameters from the functional operation.

6.1.3.7 System implementation

The XFlow application is the workflow enactment service in PRONEGI, but its purpose is also to provide an intuitive graphical user interface for modeling processes and for administering the WfBB. At first, because the WfBB had been implemented with Java, there was an attempt to build this application using the same technology. But soon it was realized that an implementation in native code would be preferable. A platform-specific application would provide a better user experience, and

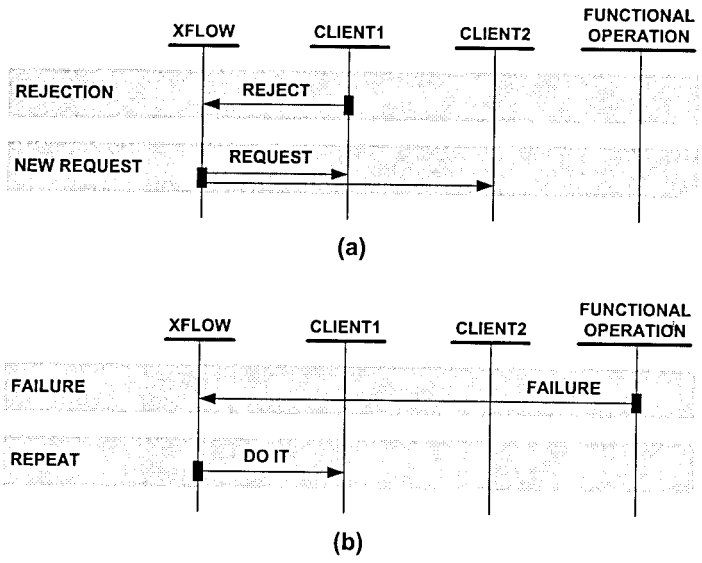


Figure 6.21: Activity rejection (a) and failure (b)

would make it easier to implement advanced user interface features[110]; it would also allow XFlow to be integrated with common desktop applications, although this last goal has not been pursued further. There was also some experience with Visual C++, which encouraged an implementation based on the Windows platform.

Integrating XFlow with the WfBB should be straightforward, since all WfBB components were accessible via CORBA. To ensure that this would be so, the first step was to devise simple command-line applications in C++ to interact with the WfBB. These applications required a C++ ORB, so the chosen ORB was ORBacus[111], which was one of the most reliable ORB implementations. Apart from some minor issues, it was possible to implement applications in C++ that were fully interoperable with the WfBB. This has demystified any fears of developing two applications with very different technologies.

As a result, the XFlow application has been developed with Visual C++, using the Microsoft Foundation Classes (MFC) to implement Windows-specific user interface features including drag-and-drop. This application has been integrated with the WfBB via CORBA, using ORBacus for that purpose.

Regarding the workflow clients shown in figure 6.19 on page 301, version (a) has been implemented in Java with JavaORB, whereas version (b) has been implemented in C++ with ORBacus.

Initial development began with ORBacus version 3, afterwards changed to version 4.0b2, and then to version 4.0.5. Still, there were some threading problems because it was difficult to conciliate the MFC threading model with that of ORBacus. These problems used to occur when XFlow Subscribers received Subjects since, in this callback situation, XFlow must invoke MFC objects from within an ORB thread, which is not a thread-safe operation.

This problem could only be solved by completely decoupling MFC code from ORB code. This was achieved as illustrated in figure 6.22. Whenever the WfBB has a Subject to deliver to one of

[110] By that time, Java Swing was still embryonic and the AWT was too heavyweight.

[111] At the time available at http://www.ooc.com/. Now available at http://www.iona.com/products/orbacus_home.htm

XFlow's Subscribers, it calls that Subscriber's receive() method. The Subscriber delegates the method call to an Observer object implemented by XFlow. In this callback situation, the WfBB is playing the role of a CORBA client, whereas XFlow is playing the role of a CORBA server. The first thing the Observer does is to retrieve three elements from receive()'s input parameters: the Subject name, the Subject type, and the name of the Subscriber to whom replies may have to be sent later on.

Then, the Observer object allocates a data structure dynamically on the heap, and fills this data structure with those data. This data structure is a neutral object because it expresses information as text strings, without any reference to CORBA objects. Having a pointer to this neutral object, the Observer sends this pointer to XFlow's main window queue. In fact, the Observer sends a user-defined Windows message to XFlow's main window queue, carrying the pointer as a parameter. This message is sent with PostMessage()[112], which returns immediately, so the Observer finishes its job without having to wait for the window to handle the message. Thus, the message is delivered asynchronously to XFlow's main window.

Once it arrives in that window's message queue, the message is handled according to typical mechanisms of Windows. However, XFlow implements a special procedure for handling this kind of message: as soon as the message is picked from the message queue, it is passed on to the workflow execution routines, which will decide what to do with it depending on the Subject name. For example, if the Subject name contains "SUCCESS" then the message means that a certain activity has been successfully completed. In general, XFlow decomposes the Subject name into its message, activity, and data portions and, from the activity's unique identifier, XFlow is able to find the workflow process containing that activity. Then, XFlow updates the activity's execution status and proceeds with process execution.

6.2 The DAMASCOS project

The scope of the DAMASCOS project (Dynamic Forecast for Master Production Planning with Stock and Capacity Constraints) [J. Ferreira, 1999] is very different from that of PRONEGI but, curiously enough, they share a similar technological solution. The DAMASCOS project (numbered IST-1999-11850) was partly funded by the European Commission under Key Action II (New Methods of Work and Electronic Commerce) in the Information Society Technology (IST) programme. Its goal is to develop a set of software modules that provide some specific supply chain management capabilities. The system has been devised especially for small- and medium-sized enterprises (SMEs) which produce

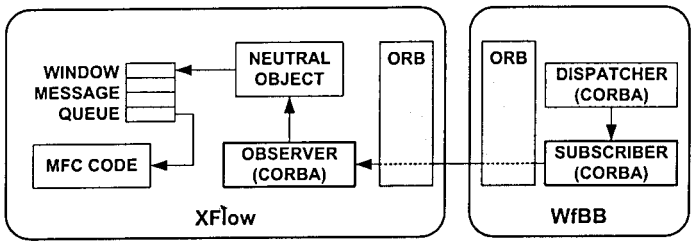


Figure 6.22: Decoupling MFC code and CORBA code

[112]Documentation for this Windows API function is available in the MSDN library (http://msdn.microsoft.com/).

consumer goods, and it aims at integrating these product manufacturers with both their customers and suppliers.

The DAMASCOS project assumes that, for these SMEs, managing a supply chain requires coordinating the actions of several entities, including raw material suppliers, internal production, retailers and sales personnel. Furthermore, DAMASCOS assumes that this coordination takes place at two different levels - the production level and the sales level - and that both of these kinds of activities are typically initiated and coordinated by the product manufacturer, which is referred to as the *principal producer*.

For many product manufacturers, a typical order is not a simple matter of supplying a ready-made product. Instead, a principal producer may have to buy raw materials from several suppliers anywhere in the world, and these raw materials must be transformed into the final product according to a certain production process, which may be different for each kind of product. Thus, argues DAMASCOS, the principal producer is the central point in a supply network, which may have to be customized for each kind of product. DAMASCOS refers to this supply network as a *customized supply network.*

The goal of DAMASCOS is to develop an open IT platform that supports production and sales activities within a customized supply network. This platform contains a set of software modules that address sales integration, inventory management and distribution, customer demand and sales forecasting, and production order management. In general, there will be a separate software module for each of these functionalities. These software modules are not intended to replace existing information systems; instead, they complement these legacy systems and interact with them in order to provide the desired functionality. The set of software modules should make up an open platform, so that additional modules can be plugged in as necessary.

Each software module has a different purpose and it may be developed independently of other modules. Thus, this modular approach requires an integration infrastructure which enables different software modules to interoperate with each other, making up a cohesive platform. This integration infrastructure should be more than a message-passing system: it should allow all information exchange between modules to be quickly configured and reconfigured at run-time, without requiring any change to existing modules. To ensure maximum flexibility, this integration infrastructure should be workflow-enabled, i.e., information exchange should be freely configured as arbitrary workflow processes, where software modules play the role of workflow participants.

The set of software modules developed within DAMASCOS, including the workflow-enabled integration infrastructure, is referred to as the DAMASCOS Suite. The DAMASCOS project consortium comprises a set of part of partners which developed the DAMASCOS Suite - three research institutions, one IT solutions provider, and one marketing research company - and two production companies where the DAMASCOS Suite was installed:

- INESC Porto (Instituto de Engenharia de Sistemas e Computadores do Porto) is a private, non-profit organization, which is an interface between the academic world and the Information Technology, Telecommunications and Electronics industrial sectors, carrying out scientific research and development, technology transfer and advanced professional training. Within DAMASCOS, INESC Porto was responsible for the project's management, as well as for the design and implementation of an workflow-enabled integration infrastructure.

- SICS (Swedish Institute of Computer Science) is a non-profit research foundation, whose mission is to contribute to the competitive strength of Swedish industry by conducting advanced and focused research in strategic areas of computer science, and actively promoting industrial

use of new research ideas and results in industry and society at large. Within DAMASCOS, SICS was responsible for devising and implementing sales forecasting algorithms.

- UFSC (Universidade Federal de Santa Catarina) is a public university located in Florianópolis, Brazil. From within the mechanical engineering department, the Intelligent Manufacturing System Group (GSIGMA) works on consulting and training services as well as research on industrial automation, shop-floor control, manufacturing integration, virtual enterprises, and other related areas. Within DAMASCOS, GSIGMA has developed a supervision module for supply networks.

- ParaRede is an IT solutions provider based in Lisbon, with vast expertise on generic software development, systems integration, networking, and e-commerce. ParaRede develops integrated IT solutions for its customers, and supplies a workflow management system of its own (described earlier in section 3.4.2). Within DAMASCOS, ParaRede made use of its extensive experience in order to develop the sales, distribution and production modules.

- NMA (Nordisk Media Analys) is a Swedish company specialized in *brand tracking*, i.e., in making continuous measurements in order to follow up and analyze the impact of marketing investments. NMA studies the communicative effect of sales campaigns regarding its size, timing and media strategy. Its services focus on providing a continuous measure of how much the market is aware of a given brand (*brand awareness*). This method has been incorporated into DAMASCOS sales forecasting algorithms.

- Ateca, located near Bologna, Italy, is a textile company responsible for several fashion trademarks, which have been rapidly expanding throughout the European market. Ateca is responsible for the styling, design, production, distribution and sales of registered trademarks, including AVIREX, C17, and EDWIN. Its goal is to implement new methods of work that can sustain its growth. Ateca was the first company where the DAMASCOS Suite was installed.

- Kyaia is a Portuguese shoe manufacturer that has been evolving rapidly. With a business strategy strongly based on exportation (90% of all production), Kyaia was able to establish its own product lines, such as Fly London, which require constant innovation. Its goal is to incorporate results from DAMASCOS in order to increase its competitiveness. Ateca was the second company where the DAMASCOS Suite was installed.

It should be noted that Ateca and Kyaia have a different supply network. Kyaia has its own production line working in-house (the factory is in fact the company headquarters) and its suppliers, located in Asia, Europe and North America, provide raw materials including leather and soles. Kyaia ships shoes mainly to retailers within Europe. For this purpose, Kyaia relies on sales agents who visit those retailers regularly. Kyaia also owns some stores which receive products directly from the factory. The Ateca supply network, on the other hand, is more comprehensive. In spite of being regarded as the principal producer, Ateca does not produce any garments by itself. Ateca subcontracts all production to manufacturing facilities located in Bulgaria, Columbia and Italy. These manufacturing facilities are Ateca's suppliers. Ateca then ships garments to its subsidiaries, to independent distributors, and to other customers across Europe via sales agents.

6.2.1 The Business Functions

The DAMASCOS Suite contains the following set of software modules:

- SALSA (Sales Integrator for Supply Chain Management Application) is intended to support sales activities, particularly those performed by sales agents, irrespective of their location. SALSA maintains a product catalog, customer information, and sales order information. Basically, it allows sales agents, which may not be permanently connected to the system, to collect orders from new or existing customers, and afterwards to synchronize that information with the remaining modules.

- IDLS (Integrated Distribution and Logistics Support System) is intended to maintain up-to-date information about the distribution of inventory throughout the supply network, i.e., information about currently allocated and maximum storage capacity for each available warehouse. The purpose of this module is to allow the principal producer to manage inventory and stock levels in order to reduce costs and delivery times.

- IPO (Interface to Production Order Management) is basically an interface module between the DAMASCOS Suite and any existing enterprise information system. It is able to import and export production orders and supplier information from and to the legacy system. It also includes some facilities that allow suppliers to remotely update the status of subcontracted production orders.

- D_3S_2 (Demand Driven Decision Support System) is the forecasting module. Its function is to support decision-making by providing sales predictions for a given period. In order to compute these predictions, D_3S_2 uses information about current sales orders, historical data about previous sales, and information about marketing investments. This last component is a key innovation in DAMASCOS.

- SC^2 (Supply Chain Smart Coordination) is a decision support system that allows the principal producer to build a graphical model of the supply network, and to simulate the behavior of the supply network on a time frame, given the real or expected execution time for each order. This module also implements some mechanisms that support the user in conciliating conflicting time requirements.

- WfBB (Workflow Backbone) is the underlying integration infrastructure that connects all these modules and supports interactions between them. The WfBB is a workflow-enabled infrastructure which orchestrates information exchange based on arbitrary workflow processes. These processes can be configured and reconfigured at run-time without requiring changes in the remaining modules.

The development of these software modules was assigned to different project partners. ParaRede was responsible for developing SALSA, IDLS and IPO; SICS developed the D_3S_2 module, whereas NMA contributed with the brand tracking mechanism incorporated in that module; UFSC implemented SC^2; and INESC Porto developed the WfBB. The first five modules contain the functionality that DAMASCOS developed in order to support supply networks. They are referred to as *business functions*. The sixth module, the WfBB, was devised as a flexible and interoperable solution to integrate those business functions.

6.2.1.1 SALSA, IDLS, and IPO

The three modules SALSA, IDLS and IPO [Rodrigues and Lopes, 2002a], which have been developed by ParaRede, have been implemented as a set of Web-based applications using Active Server Pages

(ASP) and having Microsoft Internet Information Server (IIS) as the Web server. The three modules can be accessed at three different URLs but, underneath, they share a common Microsoft SQL Server database, which stores all the data that those modules make use of. Another common feature is that not only IPO but also SALSA and IDLS have an accompanying stand-alone application, developed with Visual Basic, that allows data to be exchanged with the legacy system. For different enterprise information systems, these stand-alone applications may have to be modified, but not the SALSA, IDLS and IPO modules themselves, which can be used without modifications.

The main screen of the SALSA module contains five tabs - Catalogue, Customers, Orders, Agents, and d3s2 - as shown at the top in figure 6.23. Browsing through the Catalogue tab, the sales agent is able to choose the class of goods and then proceed to individual articles, as shown in figure 6.23. Using the menu on the left-hand side in figure 6.23 it is possible to retrieve: the customers order that have been placed for that article, the warehouses that have this article in stock, and the production orders that have been created for this article. The difference between customer orders and production orders is that each customer order collected by a sales agent may result, afterwards, in one or more production orders being placed on manufacturing facilities. It should be noted that information about warehouses is inserted into the common database via IDLS, whereas production orders are created using IPO.

Browsing through the Customers tab, SALSA allows the sales agent to retrieve information about existing customers, create new customer records, retrieve the customer orders for a given customer, or add a new customer order for that customer. As suggested by the menu on the left-hand side in figure 6.24, it is also possible to export customer information to the SC^2 module. This information is exported as an XML document and then sent through the WfBB to SC^2.

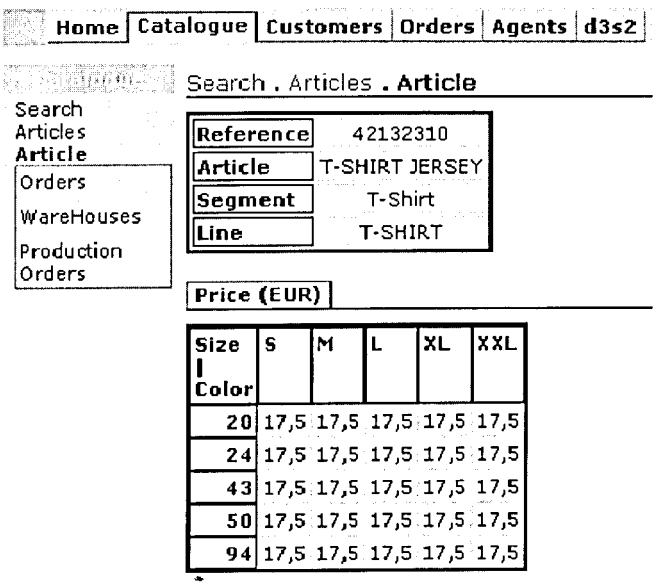


Figure 6.23: Retrieving product information using the SALSA module[113]

[113]Courtesy of ParaRede

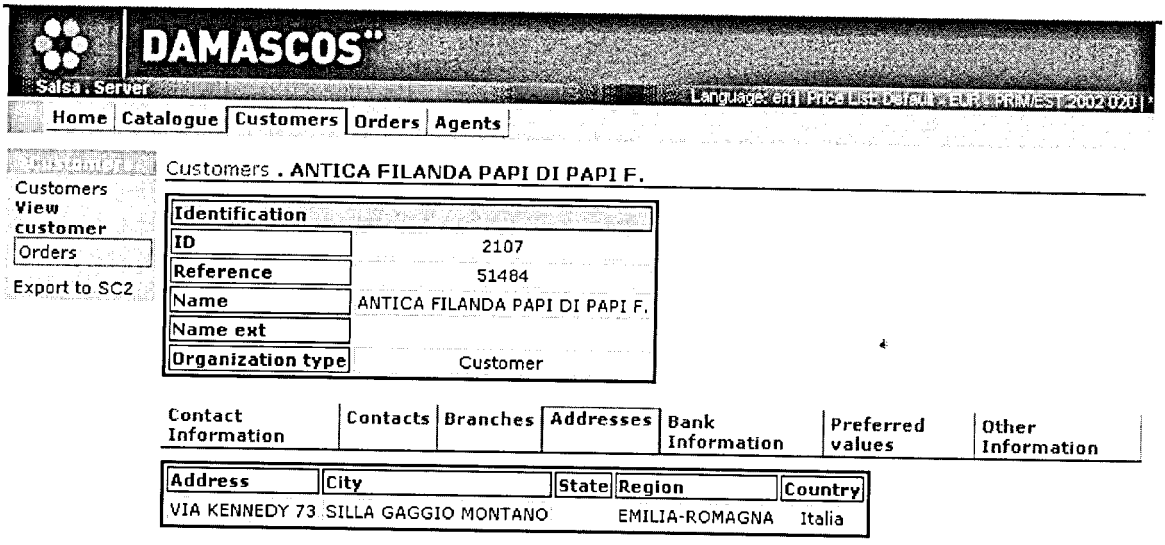


Figure 6.24: Adding customer orders with the SALSA module[114]

The Orders tab, the Agents tab and the d3s2 tab in the SALSA module are fully accessible only to the principal producer, and not to sales agents. Browsing through Orders, the principal producer is able to retrieve the orders for all customers associated with every sales agent. Customers can be associated to sales agents via the Agents page, and this page also allows the principal producer to retrieve all customer orders collected by a given sales agent. The d3s2 page is basically a front-end to interact with the D_3S_2 module: using this page, the principal producer can insert or update information about marketing investments, it can configure seasonal sales periods, and it can request sales predictions; all of these operations take place over the WfBB, as it transfers XML documents from one module to the other.

The IDLS and IPO modules have very similar Web-based user interfaces. The IDLS module allows the principal producer to specify the available distributors or warehouses, and to retrieve several details about these distributors, such as the articles that it may hold, the current stock, and the maximum capacity per article. The IPO module maintains information about production orders placed on suppliers (subcontracted manufacturing). Using IPO, it is possible to specify and retrieve information about suppliers, their production capacity, and their current production orders. Each production order has an expected start date and end date but, in practice, the real values for these dates may differ from the initial plan; IPO stores both the real and planned dates for each production order. Another feature of IPO is that it allows suppliers to update, remotely, the execution status of their production orders, as well as the real start date and end date for those orders. However, once the status of an order is set to "executed", no further changes are allowed for that order.

6.2.1.2 D_3S_2

The D_3S_2 module [Gillblad and Holst, 2002], developed by SICS and NMA, is a decision support system that is able to compute sales predictions according to two distinct scenarios: the *sales cam-*

[114]Courtesy of ParaRede

paign order scenario and the *continuous order scenario*. The sales campaign order scenario assumes that sales agents visit a pre-defined set of potential customers during a sales campaign with fixed start and end dates. In this case, sales are assumed to follow a seasonal trend, such as the Autumn/Winter or the Spring/Summer seasons for Ateca's fashion clothes. In the continuous order scenario it is assumed that orders arrive continuously at varying rate, either through sales agents that visit customers or from customers that go to the company's stores. This scenario is more appropriate for Kyaia, although there is still some seasonal behavior associated with footwear.

For the sales campaign order scenario, D_3S_2 computes sales predictions based on three kinds of data: historical data, season data, and marketing investments data. First, historical data refers to sales data from previous seasons, and it is used to compute sales dependency on time. Even if products evolve throughout the years, this data is still useful to predict sales for the same or a similar product line. It should be noted that D_3S_2 makes no predictions on single items, only on a product line or segment. Second, if there is a sales plan for the current season, then this season data will also be useful to estimate the amount of orders that agents will be able to collect. Third, with marketing investment data it is possible, using brand tracking methods, to compute an estimate of how much the general public will be aware of the brand and of the sales campaign. For the continuous order scenario, and assuming that sales still exhibit seasonal variations, D_3S_2 employs additional frequency analysis techniques.

The D_3S_2 module, which was implemented in C++, runs on a central server located at the principal producer. It was built as a forecast engine that responds to prediction requests from any module. This way, D_3S_2 does not require users to locally store large amounts of data that are required for computing predictions. The principal producer will use SALSA to submit, via the WfBB, a prediction request to D_3S_2. D_3S_2 will return the prediction values as well as the uncertainty measures so the user can judge how reliable the prediction is. Figure 6.25 illustrates the sort of result the principal producer can get from the D_3S_2 module. The SALSA module also supplies D_3S_2 with updates on new orders, marketing investments, sales campaigns, and brand tracking, whenever those data enter the system. These data are exchanged through the WfBB as XML documents.

6.2.1.3 SC^2

The SC^2 module [Rabelo and Klen, 2002], developed by UFSC, is mainly a decision support system that allows the principal producer to supervise the supply network. SC^2 receives information from SALSA, IDLS and IPO, but it does not produce any data for other modules. Instead, SC^2 provides the principal producer with a global view of the supply network. For this purpose, SC^2 allows the user to build a graphical model of the supply network, called a Smart Map, which is depicted in figure 6.26. In the Smart Map, nodes represent the principal producer, its suppliers, its sales agents, and its customers. Connections between nodes are interpreted as flow of orders between these entities. Based on information received from SALSA, IDLS and IPO, SC^2 maintains an up-to-date view of all customer orders and production orders in the supply network.

Since every order has an planned start and end date for its fulfillment, SC^2 also keeps track of the time dependencies between customer orders collected by the principal producer and production orders submitted to its suppliers. Given the real start and end dates for each production order, which are updated by suppliers via IPO, SC^2 is able to detect conflicting time requirements and to provide assistance in rescheduling orders. SC^2 has been implemented as a stand-alone application built with

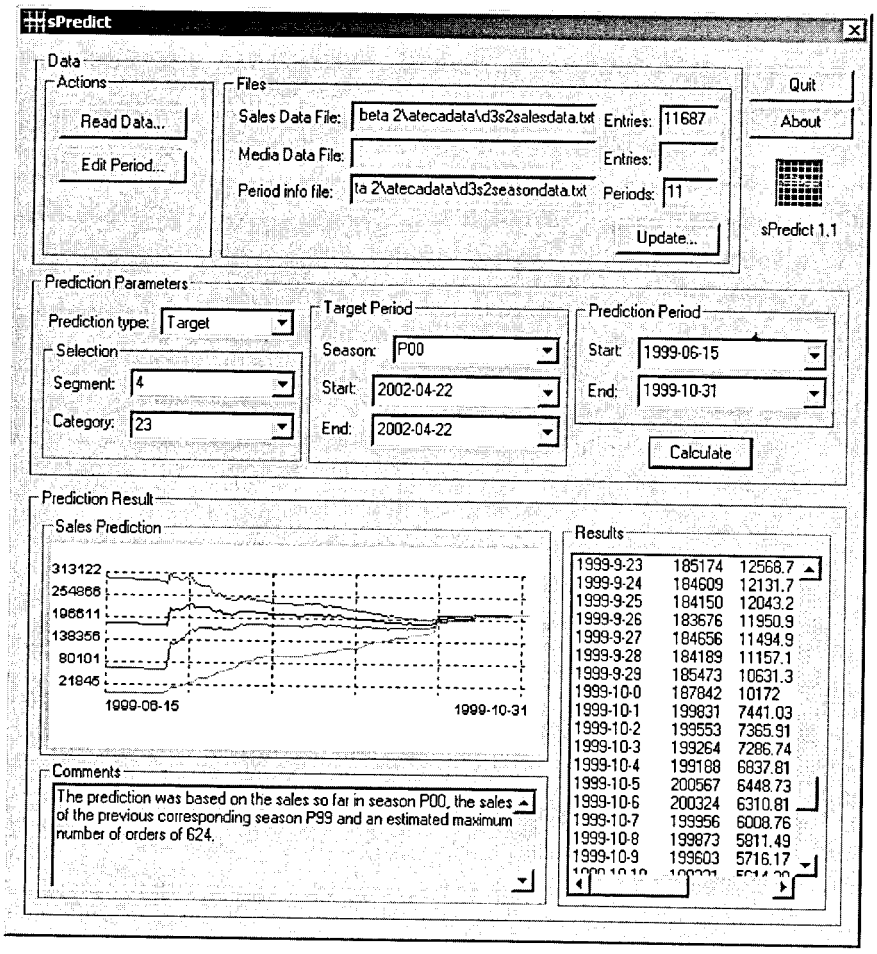


Figure 6.25: Client application used to test the D_3S_2 module[115]

Borland C++ and, for a given supply network, it is available at the principal producer's site only. Through the WfBB, it receives XML documents from SALSA, IDLS and IPO.

6.2.2 The Workflow Backbone

The DAMASCOS project aims at filling the gap between the principal producer, its customers and its suppliers within a given supply network. To achieve this, DAMASCOS makes use of two main ideas: one is that sales forecasts should drive the flow of orders across the supply network, and the other is that a flexible, workflow-enabled integration infrastructure should support information exchange between all business functions within that supply network. According to this perspective, a business function is merely an abstraction that represents any portion of self-contained functionality which is able either to produce or to consume information that flows across the integration infrastructure. Examples of business functions are SALSA, IDLS, IPO, D_3S_2, SC^2 and any other software module

[115] Courtesy of SICS. The actual user interface for the D_3S_2 module is provided by the SALSA module. The two modules communicate through the WfBB.

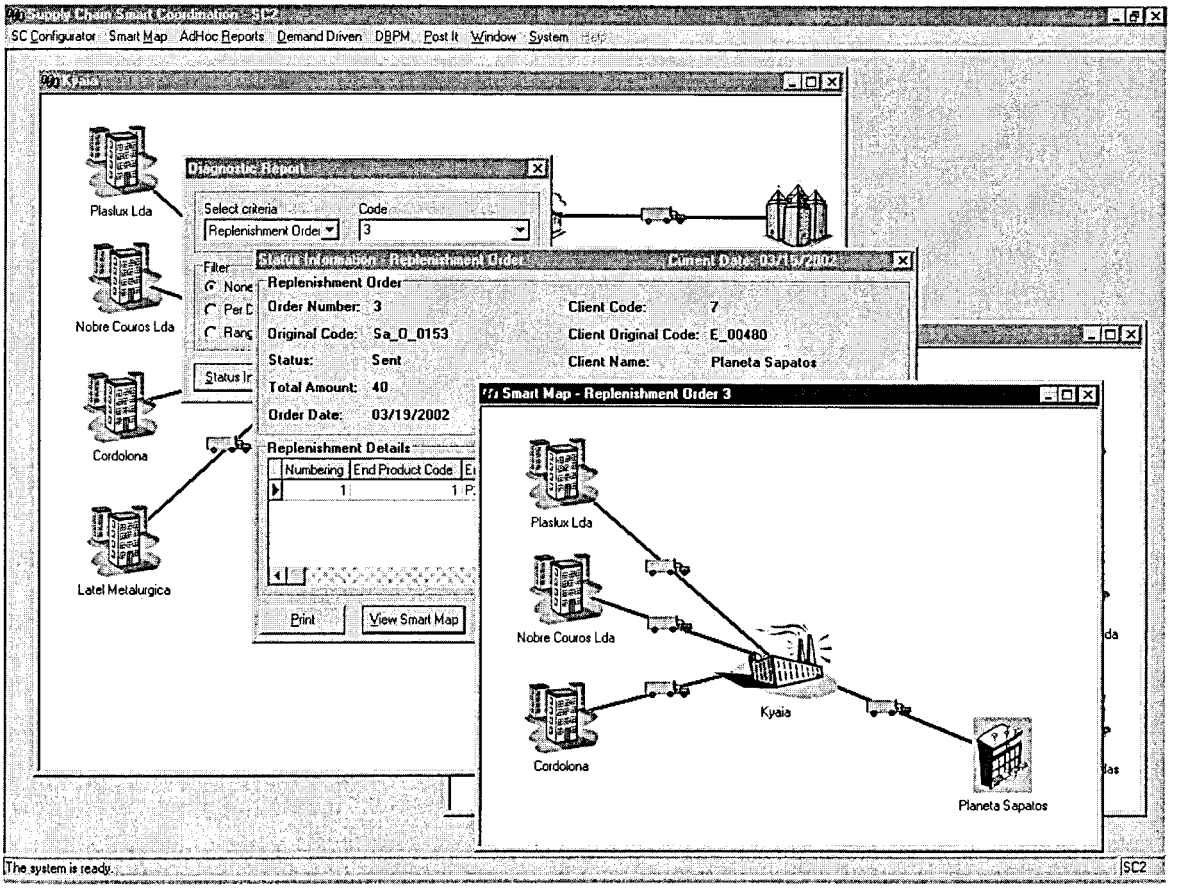


Figure 6.26: The SC^2 module[116]

that can communicate through the integration infrastructure. This conceptual architecture is illustrated in figure 6.27.

Each business function serves a different purpose and, most probably, they will be developed independently, such as what happens in the DAMASCOS Suite. One of the main goals of the integration infrastructure, therefore, is to promote interoperability between business functions. This could be achieved by means of remote procedure calls, for example. However, if this would be so, then all data exchange behavior would have to be previously coded into the business functions, and it would be impossible to use the system according to a different configuration without changing each business function. A more flexible approach would be to define, for each business function, which information it is able to provide to other modules, and which information it requires from other modules, without any assumption on the precise modules it exchanges information with. This would allow some freedom to configure data exchange behavior, and would also allow business functions to be connected or disconnected to the integration infrastructure without disrupting other modules.

Assuming that each business function is able to provide one or more types of document to other modules, and that it requires certain types of documents from those modules, then there are several document exchange possibilities. For example, SALSA is able to produce documents containing

[116]Courtesy of UFSC

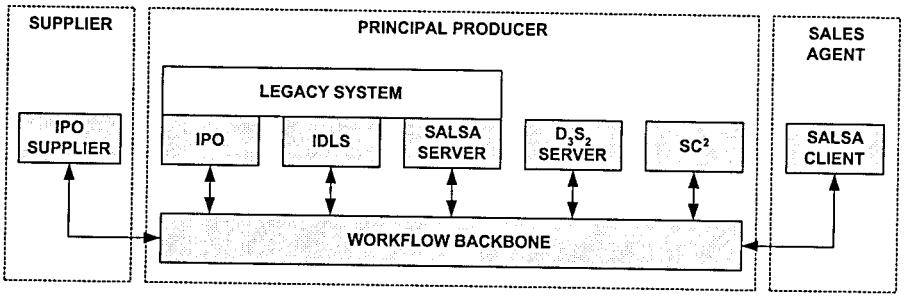


Figure 6.27: The DAMASCOS architecture

customer orders, which are required by SC^2, and another kind of documents that contain prediction requests, which is should be sent to D_3S_2. However, D_3S_2 also requires information about customer orders to compute sales predictions. All these document exchange possibilities can be gathered together in a publish-subscribe matrix [D. Ferreira and J. Ferreira, 2001], or P/S matrix, as shown in figure 6.28. Each row represents the type of documents that the corresponding business function is able to produce, whereas each column represents the type of documents that each business function requires from other modules.

The P/S matrix has been a valuable tool to identify all types of documents to be exchanged between business functions. However, it still offers a limited view of document exchange requirements. In fact, the P/S matrix does not express dependencies between document types. For example, SALSA is able to produce a prediction request document for D_3S_2, and D_3S_2 is able to produce a sales prediction document for SALSA, but the P/S matrix does not express the fact that one of these documents is produced as a response to another. In more complex cases, this chaining of documents may involve several business functions in a sequence of exchanges. Clearly, the integration infrastructure must support all these exchange possibilities. Ideally, the integration infrastructure should allow document exchange behavior to be easily configured, depending on the available business functions, and to be reconfigured at run-time if necessary.

To deal with this challenge, it was assumed that every business function performs some activities and that, in general, an activity takes a document as input and produces another document as output. For example, this is precisely what happens when D_3S_2 computes a sales prediction: it receives a prediction request document, it computes the sales prediction, and it returns a sales prediction

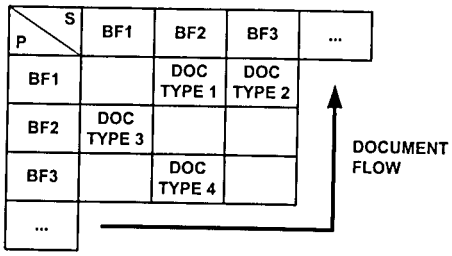


Figure 6.28: The P/S matrix[117]

[117][D. Ferreira and J. Ferreira, 2001]

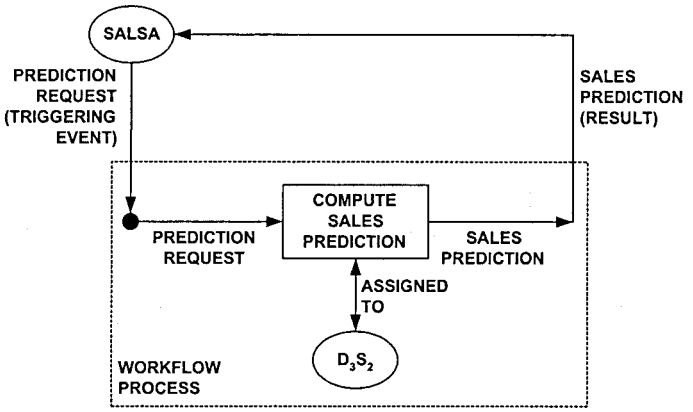


Figure 6.29: Specifying a document exchange sequence as a workflow process[118]

document. The output document may be given to another business function as an input document for its own activity, and so on, allowing an arbitrary document exchange sequence to be defined as a sequence of elementary workflow activities. In this case, the workflow process comprises a set of activities that are performed by business functions; these activities are connected by the documents that one business function produces and another takes as input. The triggering event for the workflow process is the input document for the first activity in the sequence. The simplest possible example is illustrated in figure 6.29.

As a result, the integration infrastructure is a publish-subscribe system, which makes use of workflow processes to determine, at run-time, how to route documents between business functions. This routing can be configured and reconfigured at run-time, without requiring any change to existing modules. In addition, if a new business function is added to the system, it will be able to exchange documents with other modules just by creating new workflow processes or by adding new activities to existing processes. This workflow-enabled integration infrastructure is called the Workflow Backbone (WfBB). The WfBB comprises a messaging platform [H. Ferreira, 2001], which provides message queuing capabilities, and a Workflow Facility [D. Ferreira and J. Ferreira, 2001], which provides workflow management capabilities.

6.2.2.1 The underlying messaging platform

In the DAMASCOS project, interoperability has been given a new meaning: the WfBB should provide the facilities for document exchange between a set of business functions, knowing only that those business functions would be developed independently of each other. Indeed, the WfBB was designed before the DAMASCOS business functions were designed, and it was implemented before they were implemented. Before anything else was known about business functions other than their purpose, the WfBB had already to specify the mechanisms it would implement to ensure that all business functions could interoperate.

Clearly, DAMASCOS required a reliable and robust messaging platform. In addition, this platform should make use of standard technologies whenever possible, in order to provide the maximum degree of interoperability. Developing a custom-made messaging platform such as that of PRONEGI

[118][D. Ferreira and J. Ferreira, 2001]

was simply out of the question: even the DAMASCOS project, which was considerably larger, did not have the time or resources to come up with a custom-made solution that would provide the required degree of reliability. DAMASCOS had to resort to some third-party solution, which had to employ standard technologies and had to ensure that all of the required interoperability and flexibility mechanisms could be implemented.

For these reasons, the standard CORBA Notification Service was thought to be a good solution. However, an appropriate implementation of this CORBA service had to be found. There was a Java implementation of this service proposed by OpenORB, which was used in PRONEGI, but this implementation was still in an early development stage. The TAO ORB[119] implementation of this service was too incomplete. And there was an AT&T implementation (figure 5.6 on page 175) which was also in an early development stage (but by the time the WfBB was released it was quite functional). Eventually, the chosen implementation was ORBacus Notify because it was the only implementation that was both complete and free for research purposes.

The CORBA Notification Service, as presented earlier in section 5.1.2 on page 175, includes a wide range of options for message exchange between software applications. Push and pull models, structured events, event filtering and QoS properties are some of the features that CosNotify specifies. However, the WfBB messaging platform makes use of only a subset of this functionality:

- it is possible to create several event channels, but the WfBB messaging platform uses a single event channel;
- it is possible to exchange untyped events, structured events and sequences of events, but the WfBB messaging platform makes use of structured events only;
- a structured event may carry any CORBA object, but the WfBB messaging platform uses the structured event to convey XML text documents only;
- there are forwarding filters and mapping filters, but the WfBB messaging platform makes use of forwarding filters only;
- it is possible to specify several QoS properties for channel, administration, proxy and event objects, but the WfBB messaging platform just specifies, at the channel level, persistent events, persistent connections and FIFO order policy, leaving the remaining QoS properties with their default values;
- there are synchronous and asynchronous proxy objects, but the WfBB messaging platform makes use of synchronous proxy objects only;
- it is possible for both suppliers and consumers to use the push or pull model, but the WfBB messaging platform makes suppliers use the push model, and consumers use the pull model; this means that messages are delivered using *polling* instead of *callback*, i.e., consumers must check regularly if they have new messages in their queue.

6.2.2.2 Users, user groups and message filtering

Any application that connects to the WfBB messaging platform and uses it to exchange messages is called a *user*. Every user is able to send and receive messages through the WfBB. In general, each

[119] http://www.cs.wustl.edu/~schmidt/TAO.html

business function is expected to be a user of the WfBB. SALSA, IDLS, IPO, D_3S_2 and SC^2 are all users of the WfBB, although each of these modules may have more than one WfBB user. For example, D_3S_2 has a single WfBB user, SC^2 has two WfBB users, and SALSA has one user for each sales agent. Each WfBB user corresponds to two objects in the CORBA Notification Service: one push supplier to send messages, and one pull consumer to receive messages, as shown in figure 6.30.

Each user has a unique name within the WfBB. This means that it is possible to address a user directly by means of its name, i.e., it is possible to push a message to a particular user by specifying its name as the recipient. The WfBB messaging platform also allows one single user to be specified as the *default push user*, which will receive all messages that are pushed without an explicit recipient. Another possibility is to address the message to a *user group*, so that the message will be delivered to all users within that group. An interesting feature of the WfBB is that it allows a user group to contain other user groups. A user that belongs to a given group also belongs to all groups that contain that group. The resulting tree of user groups is equivalent to the Context tree in PRONEGI.

In order to deliver a message to a specific user, to the default push user, or to a user group, the WfBB messaging platform relies on the filtering mechanisms provided by the CORBA Notification Service. On one hand, the WfBB associates one filter with the proxy object of each consumer, as shown in figure 6.30. This filter contains a set of name-value pairs, which hold the user's name and a list of all the groups the user belongs to. On the other hand, each message is a structured event which may contain a set of filterable elements in its body, according to what was shown in figure 5.11 on page 179. Whenever a message is pushed by a supplier, the WfBB automatically inserts into the message a single filterable element that contains:

- the name of the recipient user, if the message is explicitly addressed to a specific user, or
- the name of the default push user, if the message is pushed without an explicit recipient, or
- the name of the recipient group, if the message is explicitly addressed to a specific user group.

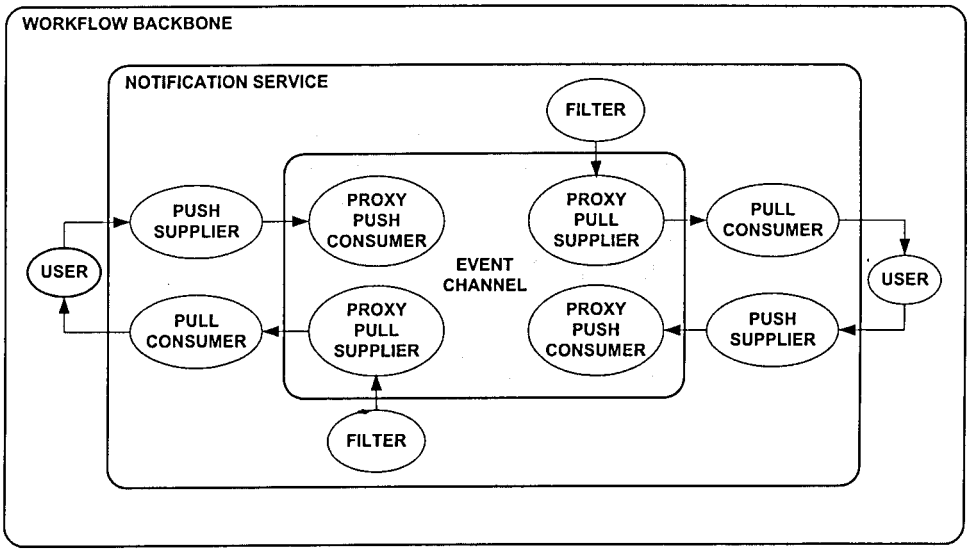


Figure 6.30: WfBB users and CosNotify objects

The result is shown in figure 6.31. The filter contains one user_id elements and zero or more group_id elements, which denote the groups the user belongs to. The message contains either a user_id element, which may denote the recipient user or the default push user, or a group_id element, which denotes the recipient group. The message will be delivered to the consumer if and only if the message element matches one of the filter's elements.

6.2.2.3 CORBA services and interfaces

In addition to the Notification Service, the WfBB messaging platform makes use of three other services: the Naming Service, the Time Service, and the Administration Service. The first two of these services are standard CORBA services, whereas the third - the Administration Service - is a specific service of the WfBB messaging platform. The Naming Service contains CORBA references to Administration Service objects. The Administration Service stores two important object references in the Naming Service, which client applications must retrieve whenever they want to connect to the WfBB messaging platform. The Time Service is used by the Workflow Facility to record the time at which certain workflow objects are created.

The WfBB Administration Service comprises a set of CORBA interfaces that allow client applications to perform two kinds of operations: administrative operations such as configuring users, and communicative operations such as pushing messages. In general, these operations are carried out by different kinds of client applications and, to regular users, only the second set of operations is available. Figure 6.32 depicts the main interfaces that allow these operations to be accessed.

The UserManager and LoginManager interfaces represent two singleton objects of the WfBB messaging platform. They are the two object references that the Administration Service registers in the Naming Service. The UserManager interface contains the methods that allow a client application to create users and user groups, as well as setting the default push user. The LoginManager interface contains just a single method that users must invoke in order to connect to the WfBB messaging platform. Each user has a name and password that it must provide before being able to exchange

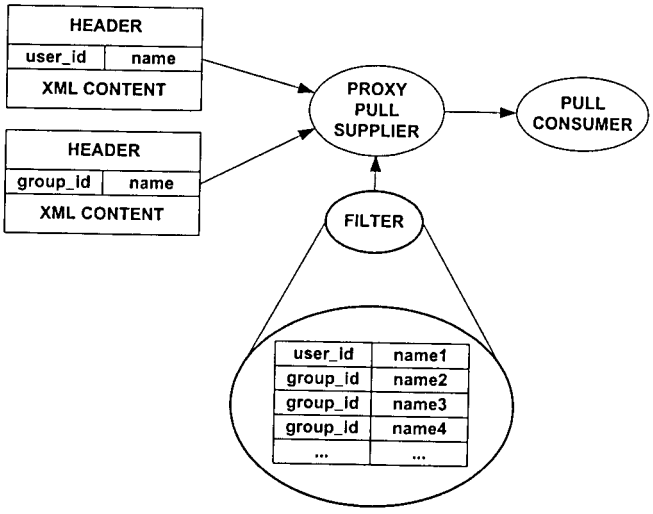


Figure 6.31: Message filtering within the WfBB messaging platform

messages with other users. The user provides these elements when it invokes LoginManager's login() method.

The login() method is an *authorization* mechanism rather than an *authentication* mechanism. The user supplies a name and password, and the WfBB verifies that the password is correct for the specified user. If the password is correct, then login() returns a reference to a User object, which represents simultaneously a push supplier and a pull consumer, as shown in figure 6.30. With this object, the user will be able to perform the following operations:

- push() - the user publishes a message (an XML document), which will be sent to the default push user;
- pull() - the user checks if it has received a new message; this method is non-blocking and returns immediately if there are no new messages; if a new message is available it is retrieved together with the name of the sender;
- pullWait() - the user checks if it has received a new message; this is a blocking method, so if there are no new messages the method waits until a new message arrives; if a new message is available it is retrieved together with the name of the sender;
- pushUser() - the user sends a message to a another user, which is specified by name;
- pushGroup() - the user sends a message to a user group, which is specified by name.

The User interface is a sub-type of the UserIdentifier interface, which gives all users and groups a unique identifier within the WfBB. Besides, it allows client applications to retrieve the parent group for a given user or user group. In fact, as suggested by the relationships between the UserIdentifier

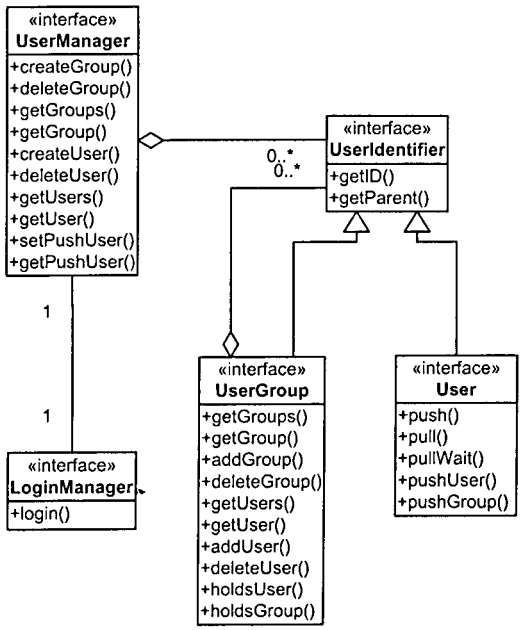


Figure 6.32: Main access interfaces to the WfBB Administration Service

and UserGroup interfaces, each UserGroup maintains a list of identifiers which refer to child users and child groups. The UserGroup interface contains all the methods required to add, remove and retrieve child elements to a specific user group, and to test whether a certain user or group belongs to the user group.

6.2.2.4 The COM interface layer

Despite the fact that the WfBB messaging platform provides a set of simple CORBA interfaces, it had been anticipated that these interfaces would still be difficult for some software modules to use. Along with the same trend that had already affected PRONEGI, as discussed in section 6.1.1.2 on page 283, it was expected that some business functions would be developed according to a Web-based client/server architecture. Moreover, due to ParaRede's company strategy, it was expected that at least SALSA, IDLS and IPO would be developed using Microsoft technologies such as ASP and IIS. At this point it was realized that, unless the WfBB would provide some ability to interoperate with these technologies, it would run the risk of being regarded as being too cumbersome to use.

The solution to this issue was to provide, along with the CORBA interfaces, a COM interface [D. Ferreira and H. Ferreira, 2001]. This COM interface is simple to use and to integrate with applications developed with Microsoft tools and technologies. Basically, it provides to COM clients a similar interface to what the WfBB messaging platform provides to CORBA clients, and it makes a bridge between the two technologies. The COM interface is a software layer that receives a COM method call and invokes the corresponding CORBA method; the result of the CORBA call is then returned to the COM client. It should be noted that, because suppliers are always push suppliers and consumers are always pull consumers, there are no callbacks and, therefore, there is no need to invoke COM methods from within CORBA calls. This simplifies the COM interface, as it needs to translate COM calls into CORBA calls, but not the other way around.

The COM interface layer implements four COM objects, as shown in figure 6.33, some of which having the same names as their CORBA counterparts (LoginManager and User). The Bootstrap object implements the necessary behavior to obtain a CORBA reference to the LoginManager singleton object, which has the interface shown in figure 6.32. The Bootstrap object has a single COM method, called getLoginManager(), which retrieves the LoginManager reference from the Naming Service using CORBA method calls. Afterwards, it creates a COM object that represents the CORBA LoginManager, which is also called LoginManager, and it stores the CORBA reference in an internal variable. If Bootstrap is unable to retrieve the LoginManager reference from the Naming Service, it does not create the LoginManager object and it returns a null reference to the COM client.

To summarize, the COM client creates a Bootstrap object, invokes its getLoginManager() method, and obtains a reference to a new COM object called LoginManager. Using this object, the COM client can call the login() method. If the login() operation is successful, the LoginManager object creates a new COM object called User. This COM object represents a CORBA interface with the same name, and makes use of a reference to a CORBA User in order to invoke communicative operations. It should be noted that the COM interface layer supports communicative operations only, not administrative operations on the WfBB.

The COM object called Message does not correspond to any CORBA object. It is just an object that represents and incoming or outgoing message. The Message object has two text parameters that convey the name of a user and the content of a message. The user's name may denote an intended recipient, if the message is being pushed, or the original sender, if it was pulled from the WfBB. The

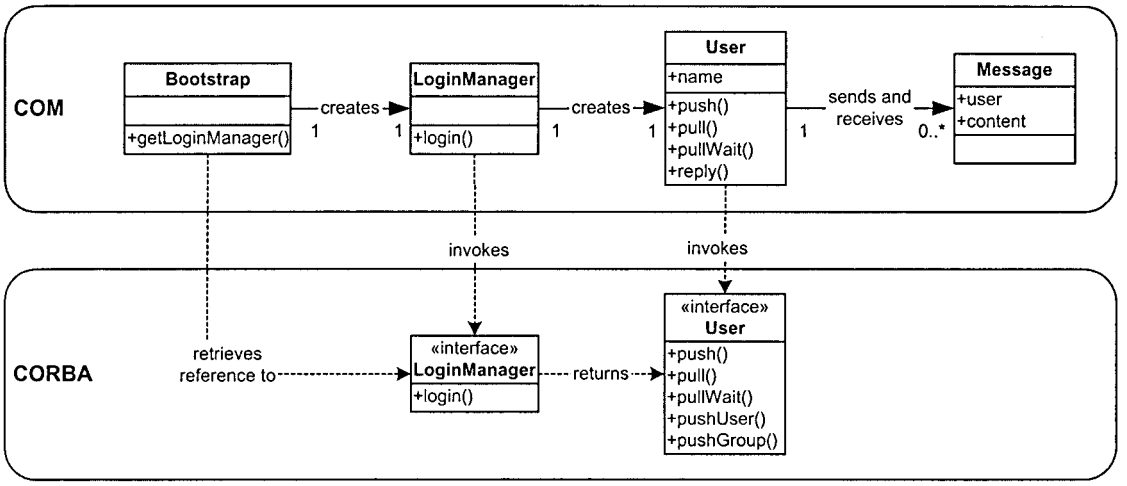


Figure 6.33: Relationships between COM and CORBA objects

message content is expected to be any stream of text. The Message object is used as input and return parameters in COM User method calls.

The COM User operations have the following relationships with the corresponding CORBA operations:

- the COM push() method, which takes a Message object as input parameter, corresponds to the CORBA push() method, i.e., it reads the message content from the Message object and sends this message to the default push user;
- the COM pull() and pullWait() methods correspond to the CORBA pull() and pullWait() methods, respectively; they both return a Message object containing the sender's name and the message content;
- the COM reply() method, which takes a Message object as input parameter, corresponds either to the CORBA pushUser() method or to the CORBA pushGroup() method, depending on whether the intended recipient is a user or a group; the COM interface layer automatically checks if the specified name is the name of a user or group, so there is no need for the COM client to specify the intended behavior.

Some implementation issues are worth noting. First, the COM interface layer has been built as a in-process COM server (a Dynamic Link Library or DLL) so that COM clients can load it into their own address space. Second, the COM components have been implemented using the free-threaded apartment model, so that COM clients can load them into an existing multi-threaded apartment (MTA) and invoke their methods without the need for marshaling [Corry et al., 1998]. Third, the COM interface layer has been implemented in C++ using the Active Template Library (ATL) [Rector and Sells, 1999]. For these reasons, the COM interface layer incurs minimal performance loss.

The SALSA, IDLS, and IPO modules have been developed as Web applications with ASP, so the COM interface layer components must be invoked from within the IIS Web Server. To make this possible, two minor configuration steps must be undertaken during the installation of those modules

[Rodrigues and Lopes, 2002b]. One is to enable Distributed COM (DCOM) on the machine running IIS. The other is to configure the DCOM *default access permissions* and *default launch permissions* in order to enable the IIS system user to run COM components.

6.2.2.5 The Workflow Facility

The Workflow Facility (WfFacility) [D. Ferreira and J. Ferreira, 2001] is an essential part of the WfBB, although it has been developed separately from the messaging platform. One of the goals of the WfBB is to provide the maximum degree of interoperability between different software modules. This same goal was kept in mind when developing the WfFacility. The WfFacility is a flexible way to configure and enforce message exchange between software modules connected to the messaging platform. But, on the other hand, interoperability also means that the WfFacility, as a workflow enactment service, should be interoperable with other workflow management systems. In fact, the kind of standard solution that was found for the messaging platform, by choosing the CORBA Notification Service, should be followed by an identical effort to devise a standard-compliant workflow enactment service.

For these reasons, it was decided that the WfFacility should be developed as an implementation of the OMG Workflow Management Facility Specification [OMG, 2000a], presented earlier in section 3.3.2 on page 77. The OMG Workflow Management Facility standard specifies the interfaces that a workflow management system should implement in order to become interoperable with other workflow management systems. In particular, it specifies these interfaces as a set of CORBA interfaces, which is quite convenient since the WfBB is a CORBA-based infrastructure. In the design of the WfBB's WfFacility, the OMG Workflow Management Facility was adopted not only as external interface, which was its original purpose, but also as the inner structure for the WfFacility. As a consequence, the class diagram shown in figure 3.9 on page 79 is the class diagram for the WfFacility, where each class corresponds to an equivalent CORBA object.

At first sight, implementing the OMG Workflow Management Facility standard seemed to be a straightforward task: the interface definitions were already given in CORBA IDL, so it would just be a matter of creating the CORBA objects and filling in the method implementations according to the OMG specification. However, the implementation of this standard would soon disclose some problems. It is worth noting that CORBA interoperability itself is all about following standards, but even then there can be different interpretations of the same standard, which sometimes leads to incompatible implementations. The implementation of the OMG Workflow Management Facility was no exception to this rule, and some issues had to be circumvented with additional mechanisms that do not belong to the original standard.

All in all, there have been five major issues that had to be solved:

- The OMG Workflow Management Facility standard specifies the syntax for the programmatic interfaces that a workflow management system should implement, but the standard does not fully address the semantics that underpin the use of those interfaces. In some cases, it is not perfectly clear what should be expected from the implementation of certain methods.

- Some design decisions that the standard contains are difficult to implement without some programming artifices. One example is the hierarchy of events, shown in figure 3.9 on page 79, which may not be necessarily translated into an identical class hierarchy in the target programming language. Another example is the hierarchy of iterators (lists of objects), which required the development of a reusable, template base class.

- The standard interfaces contain methods to find out the activities in a workflow process, but there are no methods to determine the execution order for those activities, i.e., to determine which activities come first and which activities come afterwards. To solve this problem, two new methods have been added to the WfActivity interface: these methods retrieve the previous and the following activities, respectively, for a given WfActivity object.

- The standard specifies that there are certain objects that represent resources, and that each activity can be assigned to one or more resource objects, but it does not specify any mechanism to retrieve information about all available resource objects. For this purpose, an extra object had to be design. This object is called workflow resource manager (WfResourceMgr) and it maintains a list of all WfResource objects available within the WfFacility.

- The OMG Workflow Management Facility specifies a set of interfaces that provide access to running processes and activities, but it does not specify how processes definitions should be described. In general, the OMG Workflow Management Facility does not specify how it can be used together with other standards that address essential features of workflow management systems, particularly process modeling.

6.2.2.6 Process modeling

This last challenge brought the need to adopt an additional standard in order to describe process definitions. Looking at the landscape of workflow standards, there seemed to be only one option, and that was the WPDL language, discussed earlier in section 3.3.1.1 on page 71. The consequences of choosing WPDL should not be overlooked, especially since it would become not so easy for other project partners to describe workflow processes, at least not as user-friendly as it would be desirable. Nevertheless, it was another decisive step to ensure the highest degree of interoperability between the WfBB and other workflow management systems. OfficeWorks, for example, as discussed in section 3.4.2.5 on page 100, is able to export workflow processes as WPDL files.

Incidentally, some months later the WfMC introduced an XML-based process modeling language [WfMC, 2001], which could have been a better choice, since the WfBB is also an XML document exchange infrastructure. Still, WPDL is thought to have been a good decision at the time it was made, because otherwise the only alternative would be to develop a process modeling language that would be incompatible with other workflow management systems.

Despite the choice of WPDL, and in order to improve the way processes are defined, the WfFacility has an accompanying tool, called ProcessEditor, that allows to graphically build process definitions, and to save them as WPDL text files. This tool is depicted in figure 6.34. Each activity represents a unit of work that is assigned to a specific WfBB user. This user will receive some input XML document and it will, optionally, produce another XML document. Both of these XML documents are referred to by their type.

The activity is depicted as a block with two edges: the left edge represents the input document, and the right one the output document. A red right edge represents the lack of output document. For each activity, its assigned user will receive the input document from the WfBB, and it will publish the output document, if any, in the WfBB. The WfFacility will then send the output document as input document to the following activities.

Besides defining processes, the ProcessEditor tool also allows to specify the triggering conditions for each process. A triggering condition is a tuple that contains three elements: the name of a WPDL file, the name of a WfBB user, and the an XML document type (also referred to as message type).

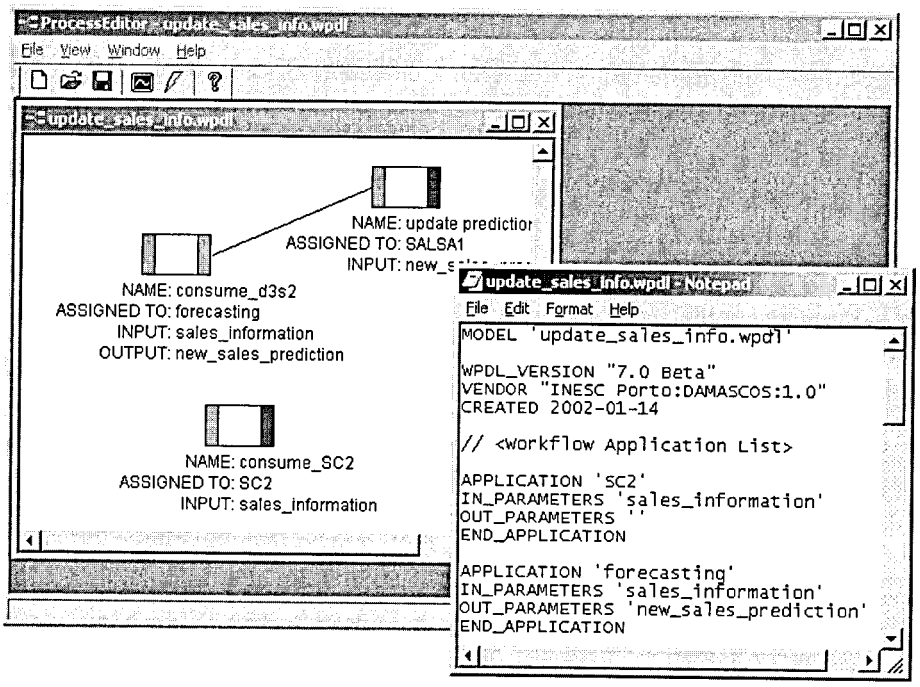


Figure 6.34: Editing WPDL files graphically with the ProcessEditor tool

The meaning is the following: whenever the specified user publishes an message of the specified type, then the specified process will be launched. The way to configure triggering conditions using the ProcessEditor tool is shown in figure 6.35. An important feature is that either the user or the message type may be specified as "any" by means of a wildcard (*). If the user is "any" then the process will be launched whenever the specified message type is received, regardless of the user who published it. If the message is "any" then the process will be launched whenever the specified user publishes any message. The process triggering conditions are saved in a special configuration file.

6.2.2.7 Process execution

The WfFacility is the WfBB's default push user. Therefore, whenever a user publishes a message in the WfBB, the message will be delivered to the WfFacility, unless the user specifies another user as the intended recipient. In general, any business function that produces a message spontaneously must publish that message in the WfBB using push(), and the message will be redirected to the WfFacility. If the business function creates a message in response to a previous message received from the WfBB, then it must reply to the original sender using pushUser() (CORBA interface) or reply() (COM interface). As a result, business functions either push messages to the default push user, or they reply to a previous sender, so they need absolutely no prior knowledge about the names of other users connected to the WfBB. If all business functions comply with these rules, then it is possible to configure all message exchange behavior centrally at the WfFacility by means of workflow processes.

Assuming that all processes and their triggering conditions have been defined, the WfFacility implements process execution by means of the following algorithm, illustrated in figure 6.36:

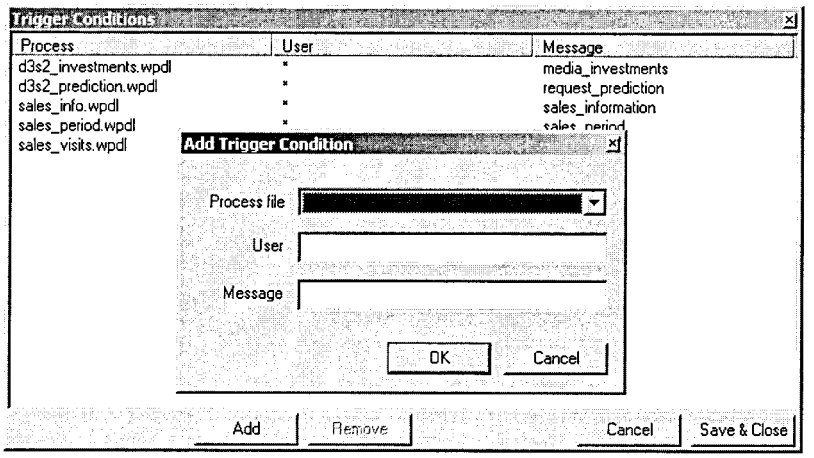


Figure 6.35: Setting process triggering conditions with the ProcessEditor tool

1. When the WfFacility starts up, the first thing it does is to read all WPDL files from the current directory. For each of these WPDL files, it creates a WfProcessMgr object, which has the same name as the corresponding WPDL file. The WfFacility stores references of all WfProcessMgr objects in the Naming Service.

2. As a second step the WfFacility creates, if it does not already exist, its own user that will listen to messages coming from the WfBB. This user is set as the default push user, and it enters a never-ending pullWait() cycle.

3. A business function publishes an XML message on the WfBB, spontaneously. This message may have been created because a sales agent has collected new customer orders, because there is a new production order, or because new data has been read from the legacy system, for example. In the example shown in figure 6.29 on page 315, a new message is created because SALSA is requesting a new sales prediction.

4. Having generated the message spontaneously, the business function publishes the message on the WfBB using push(). The message is received by the default push user, and the WfFacility determines the message type by means of the Document Type Definition (DTD) file specified in the XML document.

5. Having received a new message, the WfFacility reads a configuration file that contains the triggering conditions for all processes. Walking through the triggering conditions, the WfFacility tries to find those tuples which match the received message type and the name of the its sender. If such a match is found, the WfFacility retrieves the name of the WPDL file containing the process that should be triggered. With this file name, the WfFacility retrieves the corresponding WfProcessMgr object from the Naming Service, and requests that WfProcessMgr to create a new process instance. The fact that all WfProcessMgr objects are available from the Naming Service allows other software applications, besides the WfFacility, to create process instances as well.

6. At this point, the WfProcessMgr creates a new WfProcess object. It then reads the WPDL file and parses its contents, loading the process definition onto the newly created WfProcess object.

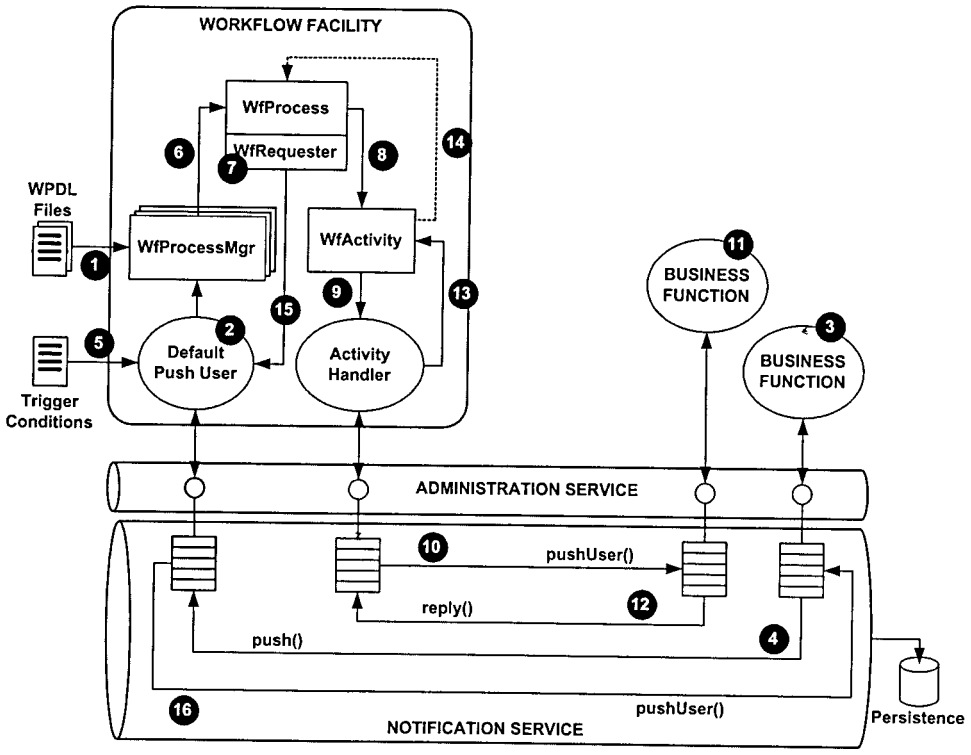


Figure 6.36: Run-time behavior of the WfFacility and the WfBB

Internally, the WfProcess object has an object-oriented data structure that stores all process definition details, such as the process activities, the input and output message for each activity, the user assigned to each activity, and the transitions between activities. After the WfProcess object has been created, the WfFacility sets the received message as the input data for that process instance.

7. When requesting the WfProcessMgr to create a new process instance, the WfFacility passes a new WfRequester object as parameter. This WfRequester object contains a reference to the user who triggered the process, i.e., the user who sent the message which triggered the process. Ultimately, when the process execution is complete, this user will receive a result message. The WfProcessMgr creates a new WfProcess object, and establishes the WfRequester as the receiver of any workflow events generated within that process instance, in accordance with the OMG Workflow Management Facility specification. These events are delivered to the WfRequester as WfEventAudit objects (not shown in figure 6.36).

8. The WfFacility starts the new process instance, and the first thing the WfProcess object does is to determine which are the first activities to be executed. As a rule of thumb, the WfProcess chooses the first activities to be those which have no connections to previous activities. Then, the WfProcess creates one WfActivity object for each of those activities. The input data for the process instance, which is the message that triggered the process instance, is given to the first activities as their input data. This is in accordance with what was shown previously in

figure 6.29 on page 315: the message that triggered the process is the input data for the first activity in that process.

9. When a WfActivity object starts, it creates a new WfBB user, which is a special user created dynamically at run-time. This user is called an *activity handler* because its sole purpose is to dispatch the activity's input data and to wait for the activity's result, if any; afterwards, this user will be erased from the WfBB. Each activity handler has a unique name.

10. The activity handler retrieves the activity's input data and the name of the user assigned to that activity. Then, the activity handler sends that input message to the assigned user, by invoking pushUser().

11. A certain business function, which that user belongs to, receives the message from the WfBB. This business function may perform some computation, such as D_3S_2 in figure 6.29, or it may update its own data, such as when SC^2 receives a status update for a certain production order. Regardless of what the business function does with the received message, it may produce some result or not, according to the original process definition. If the activity does not produce a result, the activity handler immediately sets its state to "complete", and the WfFacility proceeds to step 14.

12. If the business function produces a result, then this result is sent back as an XML message to the activity handler using reply() (if the business function uses the COM interface) or pushUser() (if the business function uses the CORBA interface).

13. The activity handler receives the XML message and sets this message as the output data for the activity. At the same time, it changes the state of the activity to "complete".

14. When the activity completes because the result was received, the WfActivity object automatically launches the following activities, if any. If the activity completes because there is no result to receive, then there can be no following activities. Regardless of the particular situation, if there are no following activities then the WfFacility proceeds to step 15. But if there are following activities, the WfActivity object creates a new WfActivity object for each of the following activities, and it sets the received result as the input data for those activities; then the WfFacility goes back to step 9.

15. If there are no more activities to perform within the process instance, the WfActivity object sets the state of the process instance to "complete". At the same time, it also sets its result message, if any, as the process result data. In response to these events, the WfProcess object notifies the WfRequester that process execution is complete. If the process has some result data, then WfRequester will request the default push user to send those data to the user who triggered the process.

16. The process result data is sent to the user who triggered the process, by invoking pushUser().

6.2.2.8 Implementation and testing

To recapitulate, the WfBB makes use of three standard CORBA services: the Notification Service, the Naming Service and the Time Service. The Notification Service, together with a special-purpose Administration Service, is the underlying messaging platform of the WfBB. The Naming Service is

where the WfBB stores references to CORBA objects that client applications will have to invoke in order to connect to the messaging platform. On the other hand, the WfFacility, which is the WfBB's workflow enactment service, also stores object references in the Naming Service that allow external applications to create process instances. In addition, the WfFacility uses the Time Service to record the time at which it creates WfProcess objects, WfActivity objects, and workflow events, according to the original OMG Workflow Management Facility specification. In fact, the WfBB makes use of three standards: the CORBA Notification Service, the OMG Workflow Management Facility, and the WPDL language to describe process definitions.

Regarding system implementation, the WfBB messaging platform requires ORBacus C++ and ORBacus Notify to run the Notification Service. The accompanying Administration Service has been developed with Java and ORBacus Java due to ease of implementation; it should be noted that this does not inflict a relevant performance loss, since most of the processing takes place at the Notification Service. The WfFacility has been implemented with C++ and the TAO ORB, with the presumed goal of developing a high-performance, reusable workflow enactment service free from licensing restrictions. It is interesting to note that the remaining software modules also employ different technologies: SALSA, IDLS and IPO have been implemented with ASP, and they use the COM interface to communicate through the WfBB; D_3S_2 has been implemented with C++ and uses the COM interface as well; SC^2 has been implemented with C++ but it uses the TAO ORB to connect via CORBA to the WfBB.

Despite the fact that the WfBB is built on distributed systems technology, the question remained as to what extent the WfBB would support message exchange between applications dispersed across a network. The WfBB could surely integrate software applications built with different technologies, but would this CORBA-based infrastructure be able to integrate applications which could be anywhere on the Internet? At first, it was thought that network devices, particularly firewalls, would make it impossible for the required network ports to be open across the Internet. But then, during testing sessions, it was found that the WfBB could indeed connect applications across wide geographical distances, as suggested in figure 6.37. In this scenario, SALSA, IDLS, and IPO were located at ParaRede headquarters in Lisbon, and the WfBB and D_3S_2 were located at INESC Porto, whereas SC^2 was at UFSC in Florianópolis, Brazil. Using instant messaging facilities, such as ICQ[120], to coordinate testing among project partners, the WfBB proved to work even when the software modules were so far apart on the Internet.

6.2.2.9 Latest improvements

During testing sessions, the number of process instances running concurrently was taken to the limits. Quite surprisingly, though, it was not too difficult to reach those limits: on one hand, the WfBB failed when the number of users reached approximately 650 users and, on the other hand, it seemed impossible to exchange XML messages with size over 500KB. Something had to be done to solve these issues, which were causing a somewhat embarrassing situation before the remaining project partners.

The reasons behind the first problem, the limit of 650 users, was found to be in the WfBB's clean-up routines and in the way the WfFacility made inefficient use of WfBB resources. In section 6.2.2.7 it was shown that, for each WfActivity object, the WfFacility creates a new *activity handler* to take care of that workflow activity. The activity handler is simply a WfBB user that is created at

[120] http://www.icq.com/

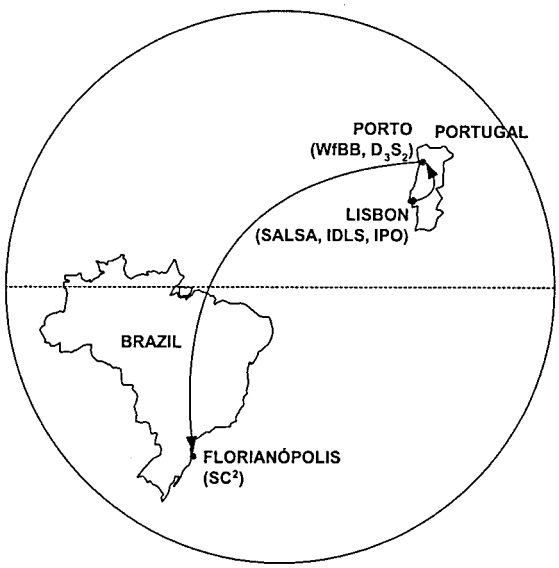


Figure 6.37: Geographical testing scenario for the DAMASCOS Suite

run-time and afterwards destroyed as soon as the activity completes. However, when the WfFacility requested the WfBB to destroy the activity handler, the WfBB was not fully deallocating all used objects. Eventually, this leak would make the WfBB run out of memory.

Two measures have been taken in order to solve this problem. One was to carefully examine the WfBB clean-up routines and to improve them so that all objects would indeed be released after use. The second measure was to change the way the WfFacility makes use of activity handlers. Instead of destroying WfBB activity handlers, the WfFacility now stores them in a user pool from where it draws an existing user whenever it needs an activity handler. If there is currently no available activity handler from the user pool, the WfFacility will create a new one and then add this user to the pool for future use. With this improvement, in a particular experiment the WfFacility required less than 20 activity handlers to deal with 1000 process instances, whereas previously it would have created and destroyed 1000 WfBB users.

The user pool has been implemented as a simple stack data structure, but access to this data structure had to be controlled in order to prevent two distinct threads from inserting or removing activity handlers into or from the pool at the same time. For example, if one thread finds that there are available activity handlers for reuse in the user pool, this thread should be allowed to obtain one of those activity handlers without another thread having meanwhile done so too; otherwise, the second thread could steal the activity handler before the first one could get it. To avoid this kind of situations to occur, access to the user pool has been protected by means of *critical sections*, so that only one thread at a time is allowed to access the user pool.

The second problem, the limit of 500KB per message, was found to be a limitation of the COM interface layer alone. COM and CORBA have different data types to deal with character strings: whereas CORBA::String is the preferred type within CORBA [Henning and Vinoski, 1999], COM recommends the use of BSTR pointers [Rector and Sells, 1999]. Therefore, in order to translate COM calls into CORBA calls, which is the purpose of the COM interface layer, there has to be some way to translate BSTR pointers into CORBA strings. The approach taken by the WfFacility was to

use an ATL string conversion macro known as OLE2A() [Rector and Sells, 1999]. This macro takes a BSTR input parameter and returns a pointer to a newly allocated ANSI string; then, it is possible to pass this pointer on to a CORBA::String object constructor.

The problem is that, internally, the OLE2A() macro uses the _alloca() routine to allocate memory for the new character array, and this function allocates memory on the program stack. Since the stack has a restricted size, it was impossible to correctly convert strings beyond a certain size limit. The solve this problem, another string conversion mechanism had to be used. The new mechanism is to use the _bstr_t object, which encapsulates the BSTR data type. First, the BSTR pointer is passed onto the _bstr_t constructor, which allocates a new string, as well as an internal character buffer, on the heap. Second, _bstr_t has an overloaded operator that provides a pointer to its internal buffer, in the form of a pointer to an ANSI string. This string is then passed onto a CORBA::String object constructor. With these modifications to the COM interface layer, it was possible to exchange messages in excess of 3MB.

6.2.2.10 Security and federation

The WfBB was developed as a research prototype, with the purpose of providing a flexible integration infrastructure to the remaining modules in the DAMASCOS Suite. In this respect, the WfBB supports all the necessary interoperability between those software applications. On the other hand, however, there was the possibility of having the DAMASCOS Suite, or at least some of its modules, installed at several enterprises. The need to integrate those modules with each other raised at least two major issues. The first major issue is federation: the ability of the WfBB to operate across the Internet was a proven fact, but what about the ability of two or more WfBB instances to interoperate with each other? The second major issue is security: what security features could the WfBB provide in order to prevent unauthorized access to the integration infrastructure?

Figure 6.38 illustrates the challenge of federating two or more WfBB instances. Basically, a user (e.g. User1) belonging to a given WfBB (WfBB1) should be able to send a message to any other user (User2) belonging to any other WfBB (WfBB2). But User1 publishes the messages it produces on WfBB1, and User2 receives messages from WfBB2 only. If User1 is to send a message to User2, then there must be some mechanism that pulls the message from WfBB1 and pushes it into WfBB2. This mechanism is called a gateway, and it makes the connection between two or more WfBB instances.

A gateway comprises one special-purpose user for each of the WfBB instances that it connects, as suggested in figure 6.39. Each of these special-purpose users belongs to its own WfBB, and it represents all remote users from other WfBB instances. This special-purpose user is called a *gateway user*. Each gateway user collects all messages, whose recipient is a remote user, from the WfBB it is belongs to. In addition, it publishes all messages, coming from remote users, in the WfBB it belongs to. The gateway users contain their own message filter just like regular users; the only difference is that their filter contains the names for all remote users. This way, whenever a local user publishes a message to a remote user, the message will be always delivered to the gateway user. It should be noted that this assumes that all users can be uniquely identified across all WfBB instances.

The job of the gateway application is then to enable message transfer between the gateway users. This can be done in two ways. One possibility is for the gateway to know which WfBB the remote user belongs to, and to send the message only to the appropriate gateway user. This mechanism requires some kind of message filtering. A second possibility is to simply broadcast the message to all gateway users in order to ensure that the intended recipient will receive it, whomever it may be.

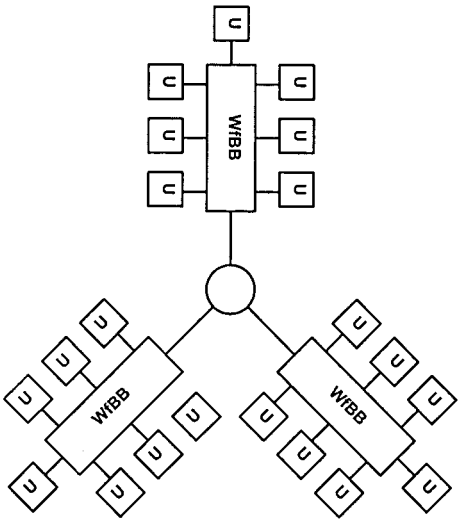


Figure 6.38: The challenge of federating several WfBB instances

Both of these mechanisms can be implemented using CORBA, or even lower-level protocols such as TCP/IP.

However, this gateway application is a potential point of failure. The gateway application uses one user to pull a message from one WfBB, and another user to push the message into another WfBB. Once the gateway application pulls the message from one WfBB, the message will no longer be available on that WfBB; and before the gateway publishes the message on the other WfBB, the message will only exist within the gateway application. If this application fails, the message is lost. Clearly, the gateway application requires some kind of transaction control over message exchange. Considering that the gateway application may also require some message filtering capabilities, then this kind of approach will ultimately lead to the need for reinventing what the CORBA Notification Service already provides.

Regarding the second challenge - security - the natural solution seemed to be the use of CORBA over SSL. The ORBacus SSL implementation was not freely available at that time, so other options had to be sought for. Alternative ORBs that implemented CORBA over SSL were OpenORB, TAO and JacORB[121]. OpenORB provided a CORBA/SSL implementation based on the Java Secure Socket

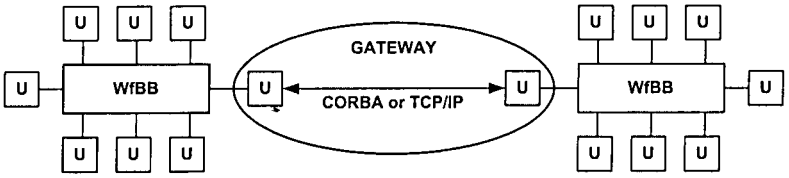


Figure 6.39: A gateway application between WfBB instances

[121] http://www.jacorb.org/

Extension (JSSE)[122], TAO's implementation was based on OpenSSL[123], and JacORB's employed the IAIK-iSaSiLk[124] SSL implementation. The first step was to create some digital certificates. Using OpenSSL, a certificate for a fictitious *certification authority* (CA) was created, together with two other certificates that would represent a CORBA server and a CORBA client. These two certificates were signed using the fictitious CA certificate.

When the connection between a simple CORBA server and a simple CORBA client was attempted, first with one-side authentication, then with authentication of both sides, all the three ORBs failed to connect with each other. This was due to mismatches in initial parameter configuration. Basically, there are several options for security capabilities, and the three ORBs had different interpretations regarding those capabilities, so they refused the configuration parameters of each other, although, in fact, their implementations were compatible. After the development teams of each ORB were contacted, OpenORB and JacORB had to be slightly adjusted, whereas TAO remained unchanged. As a result of these efforts, the three ORBs became interoperable.

Meanwhile, the use CORBA/SSL within the WfBB has not been pursued further, but two issues were already evident. One was that any of these CORBA/SSL implementations would allow only authentication and not authorization to be implemented, because the digital certificates are only used during initial connection establishment. As a result, there would be a mechanism to authenticate users, but no mechanism to determine what they would or would not be able to do. The second issue is that the WfBB would have to be changed in order to identify users based on their digital certificates rather than on an arbitrary name and password.

6.3 Findings

Both in the PRONEGI project and in the DAMASCOS project, a workflow management system has been developed with a specific purpose in mind:

- In the PRONEGI project, the purpose of the workflow management system was to control the execution of quality management procedures, which comprise several activities that are performed by human resources belonging to different departments. For each of these activities, there is a separate software application, called a functional operation, which supports the human user in performing that activity. The workflow management system assigns each activity to a user group (or Context), although only one of those users is expected to perform the activity. The activity is sent to this user together with a network address where the supporting functional operation can be found.

- In the DAMASCOS project, the purpose of the workflow management system is to orchestrate message exchange between a set of software modules developed independently. Together, these modules make up a software layer called the DAMASCOS Suite, which complements existing enterprise information systems with functionality that deals with supply chain information, namely demand forecasting, sales orders, inventory management, production orders, and supply chain supervision. Each module consumes certain message types, and produces other message types. The workflow management system decides how to transfer those messages between modules, and allows this behavior to be reconfigured at any time.

[122] http://www.java.sun.com/products/jsse/
[123] http://www.openssl.org/
[124] http://jce.iaik.tugraz.at/products/02_isasilk/index.php

Both of these systems have some features in common with other systems and approaches discussed in previous chapters:

- In PRONEGI, the Web-based functional operations can be regarded as precursors of modern enterprise portals such as mySAP.com (section 2.3.3.2 on page 55). On the other hand, the DAMASCOS Suite can be regarded as comprising a set of best-of-breed, specialized applications, as discussed in section 2.3.3.1 on page 54. For example, the D_3S_2 module can be regarded as being an APS tool, whereas the SC^2 module is comparable in purpose to SAP's APO supply chain cockpit.

- The fact that most software applications, both in PRONEGI and DAMASCOS, are built on Web technologies is a consequence of the development of 3-tier architectures such as J2EE, which favor Web-based user interfaces rather than the use of other user interface technologies, such as Staffware's GFD or Open Forms (section 3.4.1.6 on page 93).

- An important issue about Staffware is that it realizes the need for an enterprise-wide backbone, so that the workflow management system can communicate with all applications. In figure 3.16 on page 91, this enterprise-wide backbone is shown as being simply the low-level network infrastructure. In PRONEGI and DAMASCOS, the WfBB is a messaging infrastructure that promotes interoperability between software applications within the enterprise.

- The PRONEGI workflow clients (figure 6.19 on page 301), despite being quite primitive, have the same purpose of the Staffware Workqueue Manager (section 3.4.1.6 on page 93) or the Officeworks Floway front-end depicted in figure 3.21 on page 102.

- From an enterprise integration perspective, both PRONEGI's and DAMASCOS's workflow management system can be positioned within the framework of EMEIS (section 4.3 on page 150): graphical modeling tools are within the scope of model development services, workflow execution functionality is within the scope of model execution services, and the messaging infrastructure is within the scope of Shared and Base IT Services.

- In both PRONEGI and DAMASCOS, the WfBB plays a similar role to the CIMOSA Integrating Infrastructure (section 4.1.3 on page 143). An important distinction, however, is that the CIMOSA Integrating Infrastructure relies on the concept of having a global database schema, whereas the integration infrastructure in PRONEGI and DAMASCOS is event-driven. In this respect, the WfBB is more akin to the CIM-BIOSYS integration infrastructure [Weston and Coutts, 1994], which is based on a messaging platform. Many workflow management systems, such as EvE for example (section 3.5.1.3 on page 105), are designed as event-driven systems. PRONEGI's WfBB, in particular, has been designed having the observer pattern as the main inspiration.

- It is interesting to note how the observer pattern shows up, now and then, in workflow management systems. For example, in MENTOR (section 3.5.1.6 on page 107), a workflow engine is able to notify other engines whenever it initiates or finishes the execution of a workflow activity; in METEOR (section 3.5.1.9 on page 113), the task managers are observers of workflow events that are generated when a user is performing its task via the Web browser; and in CrossFlow (section 3.5.3.8 on page 129), the Workflow Module (WM) provides a push interface in order to notify other modules of "observable events". The SWAP protocol defines the Observer

interface (section 3.3.3 on page 83), and the OMG Workflow Management Facility defines the WfRequester interface (figure 3.9 on page 79) as an observer of workflow events generated within one or more process instances.

- The observer pattern is also built into the Java Message Service (JMS) via the message listener interface (section 5.1.1.2 on page 172) and into the CORBA Notification Service via push consumers (figure 5.7 on page 176). However, the DAMASCOS WfBB makes use of pull consumers in order to avoid callbacks, such as what happens in PRONEGI, which may expose the system to client malfunction.
- The DAMASCOS WfBB is workflow-enabled and XML-based messaging infrastructure, which provides the ideal features to implement a B2B framework [D. Ferreira and J. Ferreira, 2001]. In general, the B2B frameworks presented in section 5.2 focus on three issues: the interaction model, the XML document formats, and the underlying infrastructure. In the DAMASCOS WfBB, the WfFacility allows any interaction model to be configured, the XML message formats can be freely defined by the applications, and the CORBA Notification Service provides a reliable and potentially secure messaging infrastructure. The main advantage is that a workflow-enabled infrastructure such as the DAMASCOS WfBB would be able to support virtually any kind of interaction model, from the complex settlement process in bolero.net (section 5.2.4.3 on page 192) to the trivial request/response and one-way messages in cXML (section 5.2.2.2 on page 188).

These remarks put PRONEGI and DAMASCOS in perspective with the material presented earlier. However, these two experiments raised also other issues that were not previously addressed. Some of these issues concern process modeling and execution; other issues concern the integration infrastructure. In fact, both projects have produced a workflow management system that was built as a combination of these two facilities. The PRONEGI project developed a proprietary integration infrastructure, and a proprietary workflow enactment service. On the other hand, the DAMASCOS WfBB resulted in the combination of a standard, reliable messaging platform with a state-of-the-art workflow modeling and execution facility. Despite this significant improvement, and despite the fact that both systems adhere to the conclusions drawn in previous chapters, it is clear that still further improvements are required in order to come up with a workflow management system that can support the engineering of business networks.

6.3.1 Resource allocation

Resource allocation is one of the issues that workflow management systems have to deal with, but each system deals with this issue in its own way. In most cases, it is assumed that the process definition already contains the name of the resource that each activity is assigned to. In other words, it is the workflow designer who specifies which resources the workflow management system should dispatch the activities to. The standard WPDL language is built on this assumption, which is symptomatic of the fact that there are no standards concerning the way activities can be assigned to resources. As a result, workflow management systems either rely on the same assumption or, if necessary, they must devise proprietary mechanisms in order to: (1) implement different activity assignment strategies and (2) take current resource load into account.

In DAMASCOS, each workflow activity is assigned to the WfBB user whose name is already in the process definition, as suggested in figure 6.40(b). But in the PRONEGI system, a solution between

the two extremes was found: the workflow management system does not take current resource load into account, but it allows workflow participants to accept or ignore activities according to their workload. As illustrated in figure 6.40(a), each activity is assigned to a specific Context (step 1). When the activity is dispatched within the Context, it is sent to all subscribers to that Context (step 2). Assuming that workflow participants will make an honest and identical assessment of their workload, each of them will decide whether to ignore the activity or to accept it. The first user that notifies acceptance to the workflow enactment service (step 3) will be assigned the activity (step4).

PRONEGI and DAMASCOS illustrate the fact that workflow management systems may require different activity assignment strategies in different application scenarios. In PRONEGI, the activity assignment strategy shown in figure 6.40(a) was adopted in order to grant human users some degree of freedom, especially since they were accustomed to perform their tasks without strict control. In addition, there are several people with the same ability to perform certain tasks, so any of the participants can be chosen arbitrarily. The wisest option seemed to be to let them decide between themselves. Ultimately, this kind of assignment is similar to role assignment, in which activities are assigned not to individual users but to any user who is able to play a certain role within the process. Such kind of assignment is used in Panta Rhei (section 3.5.2.1 on page 116), for example.

Depending on the particular application scenario, other resource allocation mechanisms may be necessary. On the other hand, the same process definition may be run with different activity assignment strategies. For example, there is nothing preventing the process depicted in figure 6.14 on page 297 from being run with the activities assigned to individual users instead of Contexts. Therefore, the process definition should be regarded as being independent of the particular resource allocation model being employed. They are, in fact, two different kinds of models: the former addresses the sequence of activities to be performed, whereas the latter addresses the sequence of interaction steps in order to assigned each activity to its ultimate resource. The relationship between these two models is illustrated in figure 6.41.

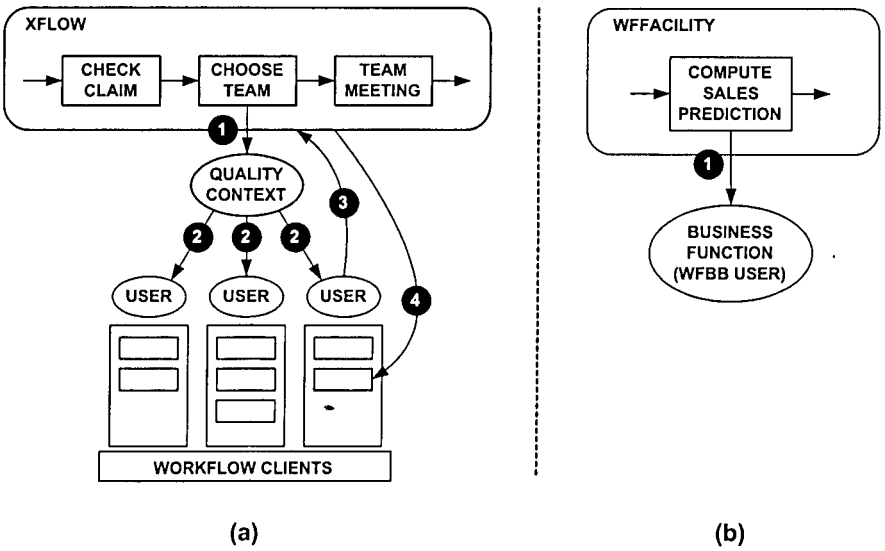


Figure 6.40: Activity assignment in PRONEGI (a) and DAMASCOS (b)

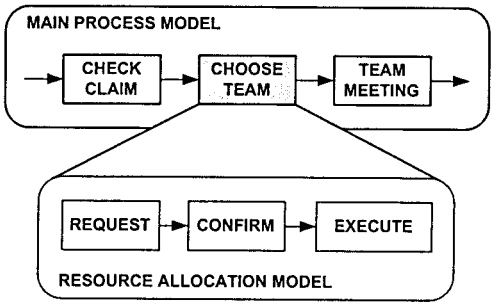


Figure 6.41: The process model and the resource allocation model

6.3.2 Model granularity

The two sorts of model depicted in figure 6.41 are independent but, at the same time, the resource allocation model can be regarded as a sub-process of the main process model. This sub-process is repeated for each activity in the main process model, because each of those activities must be allocated using the same strategy. If the resource allocation model would replace each activity in the main process model, then the main process model would appear to be much more detailed, since there would be at least three blocks for each being represented. On the other hand, this would also clutter the main process model with too much detail if one just wants to know how the process is composed, regardless of how activities are assigned to individual resources. Thus, taking the resource allocation model into account or leaving it aside actually leads to two models of the same process with different levels of detail: in the latter one sees activities as indivisible units of work, whereas in the former it can be seen that each activity requires a certain sequence of smaller steps to be executed. In this sense, the detail of a process model will be referred to as *model granularity*.

At a different scale, a similar situation may apply in inter-enterprise scenarios: one thing is the business process that an enterprise intends to run, another thing is the strategy that the enterprise employs to interact with and select potential business partners, which is required in order to perform certain business process activities. The difference from PRONEGI is that, whereas process activities were supposed to be performed by internal enterprise resources, as shown in figure 6.42(a), now they are assigned to external business partners, as suggested in figure 6.42(b). The point now is that each workflow activity, which enterprise 1 regards as being an indivisible activity, may actually be performed as a sequence of distinct activities at enterprise 2 and enterprise 3. Thus, although the three enterprises are bound to the process defined in enterprise 1, they have a different view on that process: each enterprise sees the process with different detail. Again, this is a matter of model granularity.

Model granularity is an important concept because it suggests that it is possible to have workflow processes that correspond to different levels of detail. On one hand, a process definition that describes some sequence of activities may be afterwards refined into a more detailed process model, where some activities are shown to be composed of smaller steps. On the other hand, the same process definition may represent just a sub-process within a higher-level process model. Thus, model granularity leads naturally to the concept of *process nesting*, which is basically the act of building a process definition as a structured hierarchy of processes and sub-processes.

This hierarchical view of workflow processes is essential in order to make a clear distinction between the operation environment and the engineering environment, which were introduced in chapter 4. As discussed in section 4.1.2 on page 143, the operation environment is contained within the en-

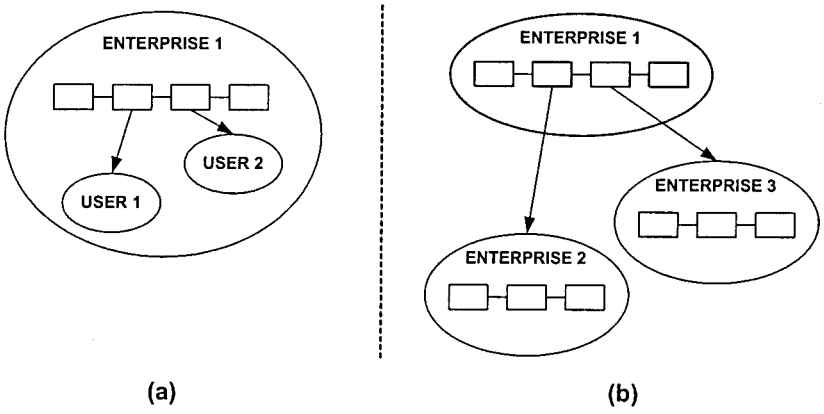


Figure 6.42: Intra-enterprise (a) and inter-enterprise (b) activity assignment

gineering environment, so an operation process is necessarily a sub-process within an engineering process, irrespective of whether this engineering process has been explicitly modeled or not.

Up to today, workflow management systems have been designed to support operation processes only. But one of the main goals of this work is to devise a workflow management system that can be helpful in the engineering environment as well. Therefore, this workflow management system must be able to deal with process definitions that concern different levels of model granularity, i.e., it must support process nesting. Many workflow management systems propose process nesting as a way to structure process definitions. But in this work, the purpose of process nesting is to ensure that at least one process modeling level is positioned at the engineering environment, whereas the remaining levels address the operation environment. This approach is illustrated in figure 6.43.

6.3.3 Activity transitions

One of the modeling features that workflow management systems usually provide is the possibility of specifying conditional transitions between activities. Typically, the transition is associated with a boolean expression whose value dictates whether the transition should be made or not. This is illustrated in figure 6.44(a), where "condition 1" and "condition 2" can be simply the opposite of each other. It should be noted that the standard WPDL language allows conditional transitions to be specified in this same way. The problem with this approach is that the conditions must be expressed as a function of data items that have resulted from previous activities. This implies that the workflow enactment service must either maintain a set of workflow relevant data or retrieve activity data from the enterprise information system, so that it can evaluate the boolean expressions. In the first case there has to be some mechanism to map workflow relevant data onto activity data, whereas in the second case there has to be a generic mechanism that allows the workflow enactment service to retrieve the activity data from virtually any kind of data source.

The XFlow workflow enactment service in PRONEGI does not support conditional transitions, but this feature would be easy to implement since the functional operations return a set of output parameters just like the input parameters shown in figure 6.16 on page 299. Given these output parameters, it would be straightforward to evaluate boolean expressions based on test values. However, these conditions would still be functions of activity data. If the functional operation misbehaves, if the

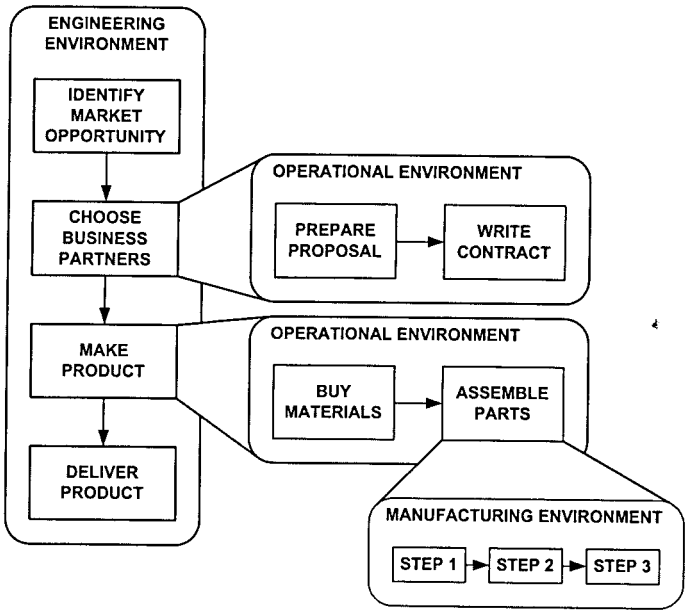


Figure 6.43: Nesting processes belonging to different environments

user makes a mistake, or if some parameter is simply missing, it is quite easy for a boolean condition to become false when it should be true. Furthermore, a boolean expression does not fully represent the fact that an activity may produce several outcomes, and that each outcome may lead the process to unfold in a different way.

A typical example of a conditional transition that shows up in many workflow processes is that of an approval. In this case, a manager is responsible for performing an activity whose result is a statement saying whether some document should be approved or not, and the process will unfold quite differently based on this decision. Most workflow management systems would represent this transition as a boolean expression in the form approved=yes or approved=no. But this solution requires the manager to know two things: (1) that a state variable called approved must be updated and

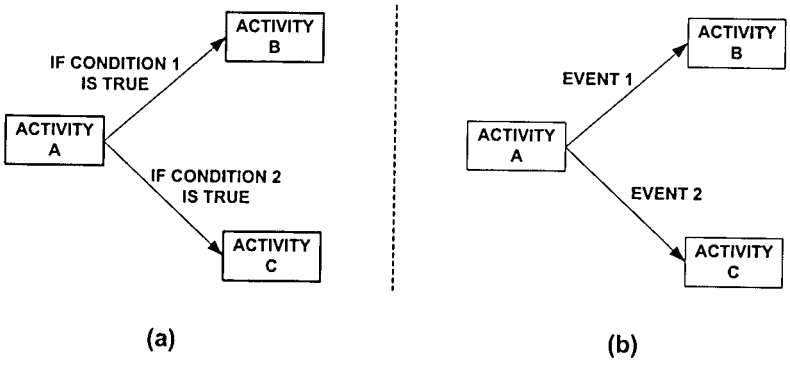


Figure 6.44: Conditional transitions based on boolean expressions (a) and events (b)

(2) that this variable takes one of two values, yes or no. A better solution would be to consider, at the time the process is being modeled, that the activity has two possible outcomes: one that represents an affirmative answer, and another the represents a negative answer. These two possibilities would be represented as two different events that the manager would generate from within its workflow client application, without having to know the name or possible values for any state variable. This approach is illustrated in figure 6.44(b).

6.3.4 Multi-task activities and ad-hoc processes

When the TR/D process shown in figure 6.14 on page 297 was being modeled, it was realized that there was a problem with the activity called "team meeting". The purpose of this activity is to identify the causes of, and decide the treatment for, a non-conformity. This is performed by a team comprising several people from different departments, depending on the sort of non-conformity. At first, it was thought that this situation could be supported by defining a new Context that would contain exactly those users, and by assigning the activity to that Context. The main problem was the fact that the team members would be selected only at run-time, by means of the activity "choose team". This would require the Context to be dynamically configured after "choose team" is performed. Since there was no satisfactory way to do this, the problem was circumvented by defining just a single activity, called "team meeting", which is performed by a single user, just like an ordinary activity. This user is given information about the team members, and it will be responsible for gathering the team together using other means such as phone calls, for example. This way, the problem of contacting team members and requesting them to attend a meeting is handed over to a human user.

The best solution to deal with this kind of challenge would be to make use of a modeling construct called a multi-task activity. The multi-task activity has been introduced by WIDE, as discussed in section 3.5.1.7 on page 111. A multi-task activity is an activity that comprises several individual tasks, which are performed in parallel. These tasks all have exactly the same purpose, the only difference being that each task is assigned to a different resource. By describing "team meeting" as a multi-task activity, it would be possible to send an individual task to each team member, notifying them that they have been selected to incorporate a team that will analyze a non-conformity. Figure 6.45 illustrates the use of this construct in PRONEGI's TR/D process.

The multi-task activity allows several instances of the same task to be sent to different resources, but then there is still the problem of specifying which resources the tasks should actually be sent to. In PRONEGI, these resources are selected at run-time, so it would be necessary to update the "team meeting" multi-task activity after the team is chosen. In other words, this means that the process cannot be defined completely until it is actually running. With a little bit more sophistication, and following the same line of thinking, it could even be proposed that several activities could be created

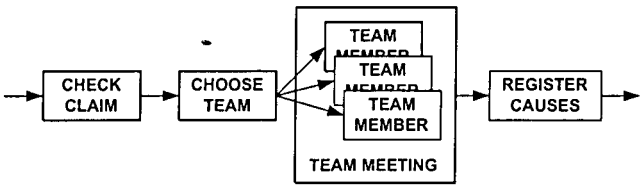


Figure 6.45: Example of a multi-task activity

at run-time, instead of using the multi-task activity construct. In this case, the process definition would be adjusted dynamically according to the results of previous activities.

Some authors have shown interest in exploring this line of thinking. For example, the ADEPT system discussed in section 3.5.1.5 on page 106 was designed to allow inserting and removing ad-hoc activities at run-time. Other authors such as [Meng et al., 2000], as discussed in section 5.3.6.2 on page 237, have proposed software agents to support processes that are run in an ad-hoc fashion. However, it should be noted that a workflow management itself is built on the assumption that processes can be modeled and afterwards executed. If, for some reason, a process or a set of process activities are only known at run-time then it probably makes more sense to look for a solution other a workflow management system in order to support those processes. Workflow management is appropriate to deal with pre-defined processes whereas software agents, for example, should be designed with the autonomy and reasoning abilities that allow them to decide, at run-time, what to do next.

Apart from this discussion, it seems to be advantageous for a workflow management system to provide access to the running process instances and to allow, to a restricted or unrestricted extent, the activities within those process instances to be adjusted or modified. In PRONEGI this is not possible, because the workflow enactment service, XFlow, is a closed application. But in DAMASCOS the WfFacility is an implementation of the OMG Workflow Management Facility specification, so each process instance is accessible via CORBA interfaces, allowing the WfFacility or an external application to reshape a running process instance by adding, removing or modifying activities.

By invoking these interfaces, it is certainly possible to replace, at run-time, an existing activity with multiple activities in order to implement something similar to the multi-task activity; and it would also be possible to create a process instance from a given process definition, and then to completely reshape that instance at run-time by building a different process, although it is doubtful whether this would be appropriate or not. Nevertheless, workflow management systems should definitely improve their flexibility; at least they should allow activities to configured or reassigned at run-time. This is an important feature for a workflow management system supporting the engineering of business networks, since the business partners that participate in operation processes have to be found at run-time, as suggested in figure 6.43.

6.3.5 Observer pattern

As noted before, the observer pattern appears to be frequently associated with workflow management systems. In PRONEGI, the observer pattern was the basis for the design of the underlying integration infrastructure; in DAMASCOS, the observer pattern is implemented both within the messaging platform and the WfFacility. What makes the observer pattern so relevant is the fact that, in essence, every workflow management system can be regarded as being an observer of events. These events make the workflow system create process instances, launch activities, change the state of activities, and proceed with process execution until it is finally complete. A workflow management system is always a kind of observer because it must be notified of events that have an impact on process execution. Ultimately, this might also explain why workflow management systems and publish-subscribe systems are often combined within a single architecture.

The observer pattern is at the very essence of a workflow activity. In DAMASCOS, for example, a workflow activity is implemented as a pair of messages: one that takes the input data to the assigned WfBB user, and another which brings back the activity's output data. This is illustrated in figure 6.46(a). Between these two steps, there is an activity handler that waits for the WfBB user to produce a response. Hence, the activity handler is an observer of the activity's result. In general, a workflow

enactment service must be an observer of external events that concern the activities that are running at some point in time.

Now, considering that an activity may be implemented as a sub-process, as suggested in section 6.3.2, the activity finishes when the sub-process completes, and the activity's output data is the output data from that sub-process. In this case, the event that indicates activity completion is the same event as, or just a sequel to, an event that signals that the sub-process has been completed. Therefore, in order to allow sub-processes to be nested within processes, not only activities must produce output events, but processes must produce output events as well. Hence, the observer pattern is also at the very essence of a workflow process. So too a process instance can be regarded as a run-time object initiated by an input event and ending with an output event, as illustrated in figure 6.46(b), while the workflow enactment service is an observer of external events that concern the process instances that are currently running.

In an inter-enterprise scenario, the observer pattern again plays an important role. A workflow activity that is to be performed at a second enterprise, either as a single activity or as a sub-process, must be initiated by an input event that crosses enterprise boundaries. Probably before, but at least when the activity is completed, the second enterprise will return a response to the first one. This response is the event that indicates the completion of the requested activity. Between these two events, the first enterprise plays the role of an observer that waits for the output produced by the second enterprise, as suggested in figure 6.46(c). It should be noted that this is the same behavior that RosettaNet assumes in its PIPs, as shown in figure 5.24 on page 202. In fact, any interaction model between enterprises can be implemented as a sequence of request-response units. Therefore, a workflow management system that supports inter-enterprise processes must implement the observer pattern within and across enterprise boundaries.

6.3.6 Integration infrastructure

Some authors suggest that inter-enterprise integration is a matter of invoking application components that dwell in another enterprise [Sachs et al., 2000]; these authors propose the use of an XML language to describe Trading Partner Agreements (TPAs), from which executable code can be generated and installed at each enterprise. Instead, in this work it is argued that inter-enterprise integration is a matter of linking and controlling business processes belonging to different enterprises, and that workflow management is the paradigm that supports this goal. In fact, workflow management is an intermediate stage between business process models and executable code: it provides a flexible way to configure

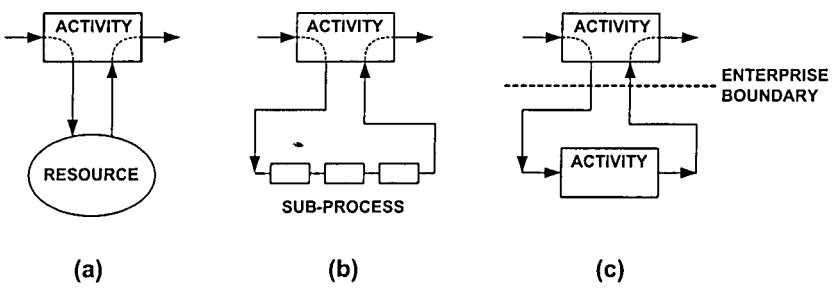


Figure 6.46: The observer pattern in different scenarios

how application components and even human resources should be invoked in order to implement a given business process.

But one of the main findings in this work is that workflow management systems require an appropriate integration infrastructure which they can rely upon, and this is why several technologies have been discussed in chapter 5. At that point, it was found that MOM systems seemed to be a good possibility. In this chapter, two workflow management systems based on messaging platforms were described. And if the purpose of this work were to come up with a workflow management system that supports internal business processes, then one of these solutions would probably suffice. But the goal is to devise a workflow management system that can support business processes in an inter-enterprise scenario. Furthermore, the goal is to support the engineering of business networks, so the workflow system should also allow previously unknown business processes to be linked at run-time.

Clearly, and despite all its advantages, a MOM system is a single, centralized bus. In PRONEGI, like in DAMASCOS, all applications must connect to a central node in order to exchange messages. If in PRONEGI this is not an issue, in DAMASCOS it becomes a relevant topic since one of the goals is to integrate the principal producer with its immediate customers and suppliers. According to the proposed solution, all software modules must connect to an integration infrastructure owned by the principal producer. For Kyaia and Ateca, the two companies where the system was installed, this was perfectly acceptable since Ateca, for example, is the owner of its suppliers and of some of its distributors. But in other scenarios it may not be acceptable to have an integration infrastructure which is under the control of a single partner.

For these reasons, it was thought that each enterprise should have its own WfBB node, so that it would have control over the integration of its own information system. At the same time, two or more business partners would be able to integrate their systems, i.e. to establish an inter-enterprise integration infrastructure, by means of connecting their WfBB instances to each other, as suggested in figure 6.38 on page 331. However, efforts to federate the WfBB could not overcome the fact that there is no reliable way to connect two WfBB instances without reinventing the capabilities that the WfBB already provides. Therefore, an inter-enterprise integration infrastructure should have similar capabilities to a MOM system, but it should definitely have a more decentralized architecture. A workflow management system that supports inter-enterprise business processes requires a decentralized integration infrastructure.

6.3.7 Electronic contracts and network-level processes

A workflow management system that supports the engineering of business networks must allow enterprises to link their business processes at run-time. This is always done with a specific purpose in mind, such as one enterprise supplying goods or rendering some service to another enterprise, and in practice this business relationship is regulated by a contract signed by all parties involved. This contract is usually called a Trading Partner Agreement (TPA) and it specifies how the purchase will take place, from initial ordering to final payment. Currently, there are proposals for TPA documents coming from two different backgrounds. One is the legal point of view that looks upon a TPA as a legal document, which binds a buyer to a supplier according to written conditions. From this point of view, a TPA should specify the ordering process details, the product items covered by the contract, the payment method, the delivery options and costs, and the performance requirements, while stipulating conditions such as privacy, warranty, and dispute resolution procedures[125].

[125] An example of such a TPA can be found at http://www.mltweb.com/ec/tpa.htm

The second point of view comes from a purely technical perspective. Recent approaches, such as IBM's tpaML (section 5.2.7.1 on page 205) and ebXML's Collaboration Protocol Agreements (section 5.2.5.5 on page 198), have led to specifications with a strong technological bias. These specifications focus on defining the message exchange sequence and the particular communication protocols used for exchanging those messages, such as HTTP, SMTP or FTP, and secure transport protocols such as SSL. However, these protocols are a concern to the underlying integration infrastructure, not the purchase process that will take place between buyer and supplier. The relevant contribution of these TPA proposals lies in the ability to specify the sequence of exchanges, which is fixed and pre-defined in some B2B frameworks.

Given that all the required technology is readily available (XML, S/MIME, HTTPS, TLS, digital signatures), the challenge is to define a contractual framework that can assist enterprises in developing legally supported business relationships. In this sense, a TPA would be a common and neutral representation for all obligations the involved parties agree on [Merz et al., 1998]. Some research projects, such as ECLIP [Canavillas and Nadal, 1999] and eLEGAL [Hassan et al., 2001], have addressed this challenge. Unfortunately, there is still little agreement concerning the contents and usage of TPAs, and there is an urgent need for standardization in this field.

From a process-based point of view, a TPA should specify the interaction model, i.e. the sequence of message exchange steps, that will take place between two or more enterprises. This interaction model comprises all the activities that are defined in one enterprise environment but must be performed in a remote enterprise environment. In this remote enterprise environment, the activity may be performed either as a single activity or as a sub-process, as suggested in figure 6.42(b). Thus, each enterprise agrees on the activities it will be responsible for, but inside the enterprise those activities may be performed according to an arbitrary business process, which the enterprise is free to define according to its own core competencies and resources.

Since the remote activities must be performed according to a certain sequence, the interaction model between two or more enterprises can be regarded as a network-level process model, as suggested in figure 6.47. This network-level process model is also known as the *public workflow*, whereas the internal business processes of each enterprise are known as *private workflows* [Aalst and Weske, 2001]. The purpose of a TPA is to define this network-level workflow process, so that the enterprises can configure their resources in order to implement this process along with their own internal processes. In this respect, the role of a TPA is much like the role of an electronic contract in CrossFlow, as suggested in figure 3.32 on page 130.

6.3.8 Security

Most of the times, and unfortunately, security is the last issue to be considered when it comes to the features of workflow management systems. PRONEGI and DAMASCOS were no exceptions to this rule:

- In PRONEGI, security issues were not considered, except for a rudimentary username/password control. This was partly because the system operates in a closed environment, and partly because the main focus was on demonstrating the use of a component- and workflow-based solution as an alternative to the monolithic, 2-tier client/server system installed at the company.

- In DAMASCOS, security features beyond password control were considered only after the first prototype of the WfBB was implemented. Efforts to incorporate SSL capabilities in the WfBB

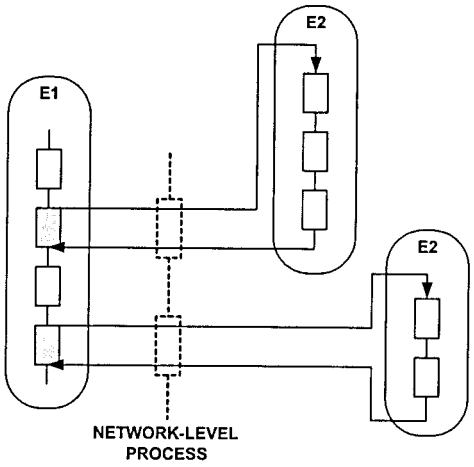


Figure 6.47: Network-level and enterprise-level processes

faced the lack of interoperability between the available ORBs. Gladly, this problem has been overcome, but eventually the WfBB remained without security improvements.

Both systems implement some mechanism to control the access to the integration infrastructure, but they do not provide secure communication channels between the applications connected to that infrastructure. In addition, the workflow enactment service in both systems does not implement any mechanism to control the access to process definitions and running process instances. In a workflow management system that is intended to support inter-enterprise processes, this situation is no longer acceptable. Either the system employs effective security measures or it will be unsuitable for inter-enterprise integration. Given that a workflow management system should be built as a combination of a workflow enactment service and an integration infrastructure, both of these components may have to implement their own security mechanisms.

Regarding the integration infrastructure, two classes of security services are required [Joshi et al., 2001]: access control services and communication security services. Access control services are intended to authenticate users before allowing them to connect to the integration infrastructure, whereas communication security services are intended to ensure confidentiality, integrity and non-repudiation of data transmitted over that infrastructure. These security requirements are often implemented with SSL and digital certificates.

It should be noted that the password control mechanisms used in PRONEGI and DAMASCOS, i.e. the login() methods, implement authorization but not authentication, since they do not check whether the username being supplied corresponds to the user issuing the request. On the other hand, as noted in section 6.2.2.10 on page 332, CORBA/SSL would allow the WfBB to implement authentication but not authorization. Thus, a combination of these mechanisms would be required in order to implement access control security. As for communication security, CORBA/SSL would provide secure communication between WfBB users and within the infrastructure itself.

Regarding the workflow enactment service, access control security is the main security issue. The question is how to ensure that only certain users have access to process definitions and running process instances, and are allowed to create, remove, or modify these objects. Such kind of access control is the focus of role-based access control (RBAC) models [Sandhu et al., 1996]. Basically,

RBAC models rely on the ability to associate permissions with pre-defined roles, which represent job functions within the organization. RBAC models are an interesting mechanism to address access control issues in workflow management systems because, in most cases, these systems already maintain some information about users and roles. In addition, RBAC models make use of role hierarchies, which some workflow management systems also support. For example, PRONEGI and DAMASCOS allow users to be arranged according to a Context tree and a user group tree, respectively.

With regard to workflow management systems, RBAC models have been proposed mainly as a mechanism to support and enforce task assignment [Joshi et al., 2001]. For example, RBAC models have been used to define authorization constraints on tasks assigned to users and roles [Bertino and Ferrari, 1999]. However, the main security requirement in a workflow enactment service is to protect process definitions and process instances from unauthorized access. But since each workflow management system is designed according to a different architecture, it is difficult to come up with an RBAC model that provides this kind of access control. Therefore, workflow management systems must usually rely on security features provided by the underlying technology or operating system.

A third security issue, besides access control and communication security, is digital signatures. A workflow management system that supports the engineering of business networks must allow business partners to establish Trading Partner Agreements, as discussed in the previous section. For these documents to have some legally binding effect, they must be signed by all business partners. Therefore, the workflow management system should be able to exchange signed documents with external business partners.

6.4 Concluding remarks

This chapter has presented two research projects in which workflow management systems played a major role. In PRONEGI, workflow management was one of the two key features in an improved information system that was built to support TQM processes. In DAMASCOS, workflow management was built into an integration infrastructure that is intended to support the exchange of supply chain information between a set of software modules developed independently. In both projects, the workflow management system has been built as a combination of a workflow enactment service and a messaging platform. But whereas in PRONEGI both of these components have been developed from the ground up as proprietary, custom-made systems, DAMASCOS was able to take advantage of at least three standards: the CORBA Notification Service, the OMG Workflow Management Facility, and the WfMC Workflow Process Definition Language.

Each of the two workflow management systems employed several technologies, and DAMASCOS has even employed technologies that are often regarded as being rival to each other. The WfBB in PRONEGI was built with Java and CORBA, while the workflow enactment service was built with C++ and CORBA. In the DAMASCOS WfBB, the messaging platform was built with a C++ implementation of the CORBA Notification Service and then augmented with an Administration Service based on Java; the workflow facility, on the other hand, was built as an implementation of the OMG Workflow Management Facility using C++ and CORBA. These systems were built to be integrated with and to integrate other software applications. In PRONEGI this was achieved with CORBA, but in DAMASCOS both CORBA and COM technologies were used. In this case, a bridge between the two technologies had to be developed.

The main point is that there is no single technology to develop such a comprehensive system as a workflow management system. Once the system has been divided into a workflow enactment service

and an integration infrastructure, then each of these components should make use of the technology that is best appropriate for its purpose. The purpose of the underlying integration infrastructure is to promote interoperability and to provide communication mechanisms between applications; the purpose of the workflow enactment service is to control the execution of workflow processes by interacting with the resources assigned to individual activities. If there would be an integration infrastructure based on WAP, for example, it would allow activities to be dispatched to mobile devices, but this does not mean that the workflow enactment service should also run on a mobile device. The challenge is not only how to implement the system, but also how to integrate the different technologies used to implement it. Ideally, these technologies should place no restrictions on the features a workflow management system is required to provide.

According to the findings described in this chapter, a workflow management system that supports the engineering of business networks should provide the following features:

- *Flexibility with regard to resource allocation models.* The workflow management system should make no assumptions about how activities will ultimately be assigned to the resources. Either the resource allocation model can be expressed as a workflow process, and in this case it can be incorporated into the process definition, or if it cannot then the resource allocation model should be left out to be implemented by external components. In any case, the workflow management system should be independent of any particular activity assignment strategy.

- *Process nesting and different levels of model granularity.* The workflow management system should allow processes to be composed of other processes, called sub-processes, and these processes to be composed of yet other processes, and so on, up to an arbitrary degree of process nesting. Process nesting allows different levels of process granularity to be represented, i.e., different views of the same process with increasing detail. The ultimate purpose of this feature is to allow operation processes to be represented as sub-processes within engineering processes.

- *Conditional transitions based on output events.* The workflow management system should be flexible with respect to activity input and output data. Therefore, the system cannot make any assumptions on the existence of state variables or on the ability of workflow participants to update data that will have a direct impact on process execution. Instead, the workflow management system can rely on workflow participants to return several kinds of events, each having a different semantic meaning. Branching decisions during process execution should be based on these events.

- *Activity reassignment at run-time.* When creating a process instance from a process definition, the workflow management system should create a set of run-time objects representing the elements contained in the process definition, particularly the activities that will be performed. During process execution, the attributes of these objects may have to be adjusted, such as the resource assigned to a certain activity. This feature is especially relevant to engineering processes, in which business partners are assigned to operation processes at run-time.

- *Observer pattern behavior across enterprise boundaries.* The observer pattern expresses a fundamental fact about workflow management systems: the fact that they behave as observers of the outputs produced by running processes and activities. Therefore, a workflow management system that supports inter-enterprise processes must implement this behavior across enterprise boundaries: after requesting an external entity to perform some activity, the workflow management system becomes an observer of the output that entity will produce.

- *Decentralized messaging capabilities.* MOM systems provide fundamental messaging capabilities that workflow management systems require in order to exchange data and events with workflow participants. However, a MOM system is a centralized infrastructure that implements those messaging capabilities at a single node. In an inter-enterprise scenario, the integration infrastructure cannot be under the control or ownership of a single entity. It must be a decentralized integration infrastructure, which allows enterprises to participate in a business network without losing their sense of autonomy.

- *Support for network-level processes.* In general, business relationships between enterprises are preceded by an agreement which specifies how the relationship will be implemented, and which binds that relationship to certain legal conditions. From the agreement, the workflow management system should be able to extract the description of a network-level process. This network-level process represents the links between business processes performed at different enterprises, and its purpose is to allow those enterprises to configure their systems in order to implement the desired behavior.

- *Access control, communication security, and digital signatures.* A workflow management system that supports inter-enterprise business processes requires an integration infrastructure which provides secure communication channels as well as access control mechanisms, including user authentication. In addition, it is assumed that trading partner agreements are digitally signed by all parties in order to ensure non-repudiation.

The next chapter describes how such a workflow management system can be designed and implemented.

Chapter 7

Proposed Solution

The previous chapters have laid out the foundation for what this chapter is about to present. From the basic premise that workflow management systems must be composed of a workflow enactment service and an integration infrastructure, to the requirements drawn in chapter 6, previous chapters have discussed the main issues and technologies that are relevant to the development of a workflow management system which is intended to support the engineering of business networks. In particular, chapter 4 concluded that such a workflow management system cannot be a centralized point of control over the actions of several enterprises, but instead it must provide the workflow functionality that, at each enterprise, will support the life cycle of business relationships with other enterprises.

The workflow management system proposed in this chapter is precisely such kind of solution. On one hand, it comprises a workflow enactment service to be installed at each enterprise, and which allows the enterprise to automate the relationships with other enterprises, as well as to coordinate the activities performed by internal resources. The key innovation factor of this workflow enactment service, which is called Workflow Kernel, is that it is a reusable core of workflow functionality, i.e., it can be used as the starting point for developing other workflow-based solutions. On the other hand, the proposed solution comprises an integration infrastructure which provides the basic services that allow enterprises to interact with each other. The key innovation factor of this integration infrastructure is that it is based on peer-to-peer networking, so it provides a completely decentralized platform that cannot be owned or controlled by any single enterprise.

The resulting integration approach is illustrated in figure 7.1. Besides the standard requirements for any integration architecture in general - that it should be open, that it should be expandable, that it should be modular, that its components should be reusable, and that it should cater for cooperative solutions - the proposed workflow management system is also a decentralized solution, both at the level of application integration and at the level of business process integration. In addition, many of its features are related to the findings described in the last chapter. As a result of ongoing investigation, it is likely that additional issues will arise, which will identify new requirements that a workflow management system supporting the engineering of business networks should satisfy. Therefore, it is not the purpose of this chapter to contain the final words with respect to this matter. Rather, its purpose is to present a possible solution, and one that is flexible enough to remain useful even in the face of future developments. Hopefully, the solution itself will suggest a possible direction to further research.

Section 7.1 describes the design and implementation of the Workflow Kernel, section 7.2 describes the design and implementation of the peer-to-peer integration infrastructure, and section 7.3 illustrates how the resulting workflow management system, which is the combination of those two

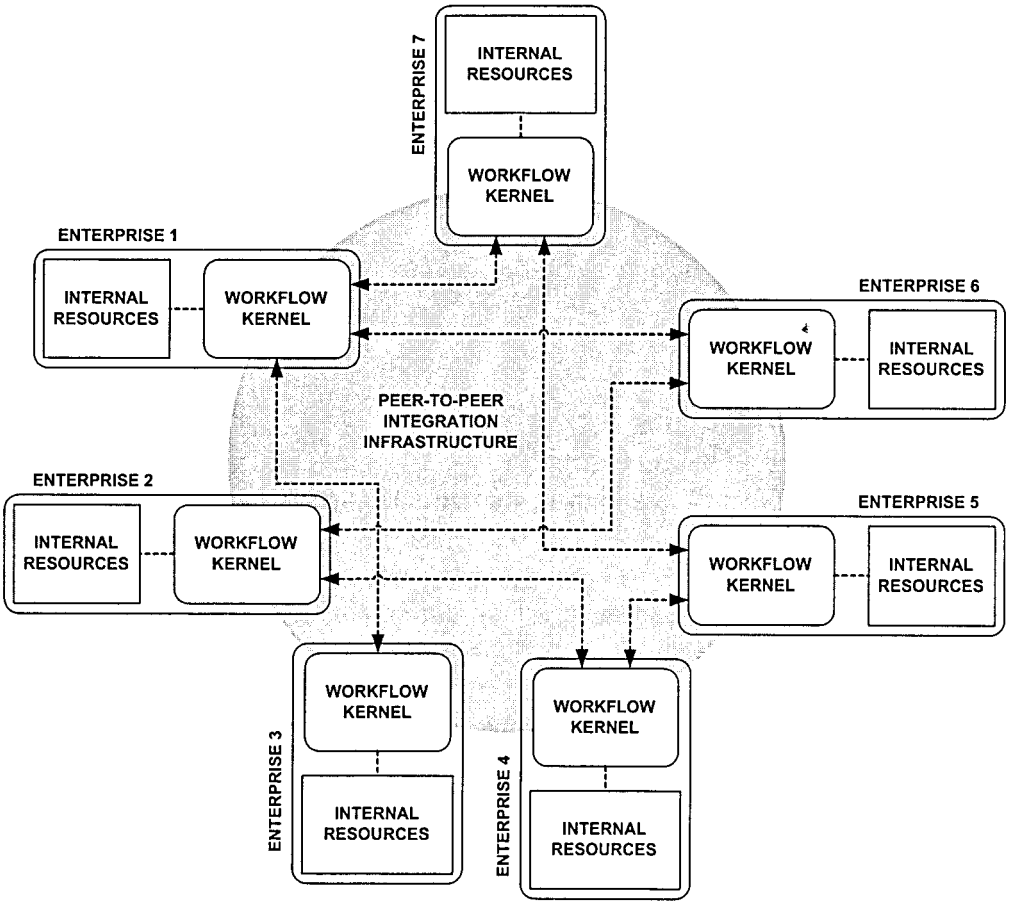


Figure 7.1: The proposed integration approach

components, can support the engineering of business networks, according to the perspectives from section 4.4.

7.1 Process modeling and execution services

Every time a workflow solution is conceived there is a large amount of functionality that is ultimately reinvented and redeveloped from the ground up. Chapter 3 presented several workflow management systems, from research prototypes to commercial applications, which have been developed to meet particular requirements. These requirements have encouraged academic researchers and commercial vendors alike to come up with their own architectures for those systems. This led to what some authors call a "scattered landscape" [Sheth et al., 1999], where numerous workflow management systems are available, based on different paradigms and offering contrasting functionality.

In fact, the WfMC's efforts to standardize the Workflow Reference Model discussed in section 3.2 on page 66 have provided a clearer picture of the scope and aim of workflow management systems, but they have not precluded developers from designing workflow management systems tailored to specific requirements. Nevertheless, the WfMC's Workflow Reference Model still remains a

useful framework to relate workflow management systems and their capabilities. Within this framework, several components interact according to a set of interfaces, as shown in figure 3.3 on page 67.

At the heart of this framework is the workflow enactment service, a process execution facility which is the run-time environment for process instances. In this sense, every workflow product, prototype or architecture entails a workflow enactment service in one way or another. Interpreting process definitions, creating process instances from those definitions, and controlling the execution of those processes are essential chores of every workflow management system, because they must be built into every workflow solution. But typically, this functionality has been implemented over and over again as each workflow management system is developed, without much regard to previous approaches or to existing standards.

This may be due to the fact the existing standards focus on the syntax of interfaces without clearly specifying the corresponding semantics and usage [Sheth et al., 1999]. Hence, when faced with specific user requirements, developers often make use of standards according to their own interpretation. This explains why workflow management systems are often incompatible, but it does not explain why developers do not take advantage, in general, of previous workflow systems and approaches.

The proposed explanation for this lack of reuse is that workflow management systems have never been regarded as a combination of two independent components: a workflow enactment service and an integration infrastructure. Still, these two components coexist in any workflow management system, but they are often so tightly coupled to each other that they cannot be regarded as two separate components, and the workflow enactment service cannot be used in a another environment. Thus, the fact that new workflow enactment services are always being developed from the ground up could be explained on the basis of two postulates:

1. none of the existing workflow enactment services can be detached from their underlying integration infrastructures, which means that a new workflow enactment service must be developed for every new application scenario;

2. when developing a new workflow enactment service, this workflow enactment service is always developed for being used with a particular integration infrastructure, so postulate 1 will apply again.

The only way to break this cycle is, for once, to develop a workflow enactment service that is independent of the underlying integration infrastructure. This workflow enactment service would then become reusable, i.e., in principle it would be possible to use the same workflow enactment service in combination with several integration infrastructures. Since a workflow management system is assumed to be, by definition, a combination of a workflow enactment service with an underlying integration infrastructure, then it would be possible to build several workflow management systems with the same workflow enactment service. In other words, it would be possible to build a workflow management system without having to redevelop those features that are common to every workflow enactment service.

7.1.1 Reusable workflow enactment services

The idea of developing reusable workflow functionality is not entirely new, as some systems have been purposefully developed with this goal in mind. For example, the Drala Workflow Engine[126]

[126] http://www.dralasoft.com/products/workflow/

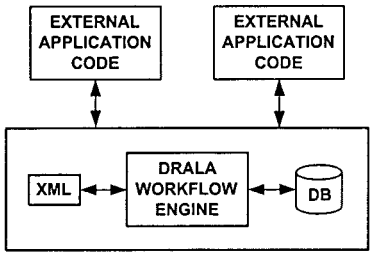


Figure 7.2: Application integration using the Drala Workflow Engine

is an embeddable Java component that provides a comprehensive API for defining, executing and monitoring processes. It is not a workflow management system by itself; it is rather a core of workflow functionality which is intended to simplify the implementation of workflow management systems. The Drala Workflow Engine already includes some support tools such as a process editor, but it allows developers to replace these tools or to build additional user interfaces. Process definitions can be imported and exported in an XML format, and the Drala Workflow Engine supports the exchange of XML data between software applications built on top of the same engine, as suggested in figure 7.2.

As a second example, the "wftk"[127], or workflow toolkit, is an open-source project that is developing a function library (in ANSI C) to provide Web-based applications with workflow behavior. The wftk toolkit has two main components: the workflow core, which controls process and activity execution, and the repository manager, which stores and retrieves data for workflow activities. As suggested in figure 7.3, both of these components rely on *adaptors* in order to interact with external systems, including databases, Web servers and third-party data formats such as process definition languages. Basically, an adaptor translates an external data format into a format that wftk is able to handle. Internally, the wftk makes use of XML-based data structures.

As a third example, the WorkMovr API[128] is a complete, Java-based programming framework for workflow management systems. In fact, WorkMovr is a workflow management system by itself,

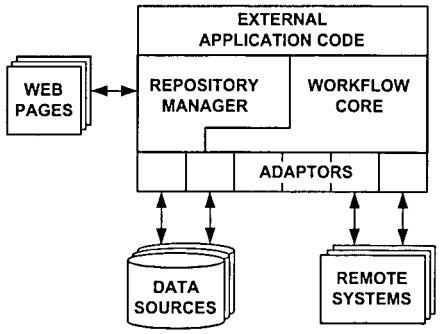


Figure 7.3: Architecture for the wftk toolkit

[127] http://www.vivtek.com/wftk/
[128] http://www.a-frame.com/

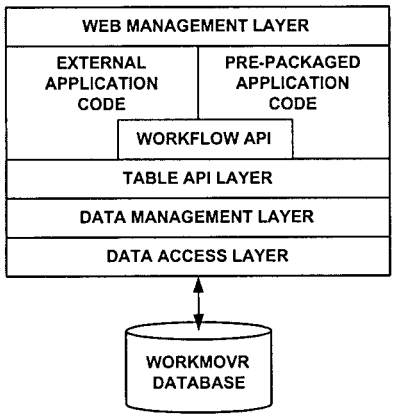


Figure 7.4: The WorkMovr layered API

but the whole of its functionality is accessible via external interfaces, which is called the WorkMovr API. The WorkMovr architecture has five layers, as shown in figure 7.4: the Data Access Layer is the closest to the underlying database system, the Data Management Layer implements enhanced data manipulation routines, the Table API hides the details of the particular database schema, the Workflow API provides access to the internal workflow engine, and the Web Management Layer implements Web-based user interfaces. Each layer incorporates and builds upon the functionality of the preceding layer. The key feature is that it is possible to develop external applications that interface with the WorkMovr system at any of these layers.

However, these system still provide a very limited degree of reusability. All of these proposals have been devised while assuming a certain workflow management system architecture, and they have basically implemented part or all of this architecture. As a result, these systems are only suitable for scenarios which their architecture fits into. Therefore, they are not fundamentally different from any other workflow management system; their advantage lies solely in the fact that they provide more open mechanisms to interoperate with external applications. A workflow enactment service that is intended to be reusable must be designed regardless of any particular workflow management system architecture. Its only assumption must be that there are some features that are common to all workflow management systems, and its purpose is to identify and implement these features. The ultimate goal is to be able to embed this workflow enactment service in any workflow solution, regardless of its particular architecture.

7.1.2 Process modeling and workflow analysis

A fundamental feature of any workflow management system is the ability to define business processes and to control their execution based on those process definitions. In section 3.4 on page 87, it was found that process modeling is one of the two typical starting points for developing a workflow management system (the other starting point being the underlying integration infrastructure). This explains why, in practice, many workflow management systems are highly dependent on their modeling approach, as some authors point out [Alonso et al., 1997]. An illustrative example is that of ActionWorkflow [Medina-Mora et al., 1993], which defines business processes as a set of four-step workflow loops, and which has been built to support the execution of this kind of models.

Although ActionWorkflow has a solid foundation on speech-act theory, the same does not usually happen with most workflow management systems which, in general, propose their own modeling languages without any formal basis [Sheth et al., 1999]. As a result, it is impossible to ensure that those modeling languages will always produce consistent process definitions. For simple processes this may not be an issue, but as processes grow in size and complexity it may not be obvious to predict the resulting behavior. Many modeling languages also introduce modeling constructs to deal with particular situations such as branching and synchronization. These modeling constructs might be inconsistent with each other, and they may not cover the entire range of possible behavior. Even if they do, it may impossible to prove it, since they lack formal semantics.

Another limitation of workflow management systems is their weak support for analysis, testing, and debugging of process definitions [Georgakopoulos et al., 1995]. This involves process verification [Sheth et al., 1999], whose purpose is to ensure that a process definition is consistent prior to releasing it for execution. The lack of workflow verification mechanisms is related to the lack of formal semantics, since the use of a proprietary modeling language requires workflow verification to be based on proprietary mechanisms as well, but these mechanisms are difficult to implement if there is no formal background. Even if they can be implemented, this will enlarge the gap between workflow management systems, because each one of them will make use of its own proprietary techniques.

A workflow enactment service that is intended to be reusable must make use of formal semantics in order to specify processes unambiguously and consistently, and to allow rigorous workflow verification techniques to be employed. The main advantage is that if the workflow enactment service provides formal semantics then all workflow management systems based on this workflow enactment service will be able to use those semantics instead of developing proprietary modeling languages. In addition, it will be possible to employ the analysis techniques provided by that formalism instead of developing proprietary workflow analysis techniques. Fortunately, such a formalism already exists, and it is known as Petri net theory [David and Alla, 1992].

The use of Petri nets for process modeling has several advantages [Aalst, 1998]:

- Petri nets have formal semantics to describe workflow processes in a clear and precise way, and they also have an intuitive, graphical nature which is easy for end-users to grasp. Petri nets include basic constructs which can be used as building blocks to specify process definitions, and they include also a set of enhancements, such as color and time, which have a formal representation. Furthermore, Petri nets can even be used as the basis to define higher-level process modeling languages [Aalst et al., 2000], if necessary.

- Petri nets have a solid mathematical foundation supported by decades of research. Several properties of Petri nets have been investigated, and many analysis techniques are available. These techniques can be employed to check process models for inconsistencies, such as deadlocks or infinite loops, and to compute performance measures, such as execution times and resource utilization. This way it is possible to evaluate alternative process models.

- Petri nets are a vendor-independent formalism to describe and analyze workflow processes. They are not based on a specific product or technology, and they are not affected by product changes or upgrades. Besides, information on Petri nets is available from independent sources, and research on Petri nets will proceed regardless of any particular workflow management system.

7.1.2.1 The soundness property

A Petri net (PN) has two types of building blocks: *places* and *transitions*. A place is represent by a circle, and a transition is represented by a box or bar. Places and transitions are connected by means of *arcs*. Arcs are represented by arrows, and they connect either a place to a transition or a transition to a place, as shown in figure 7.5. Every PN has a finite number of places, transitions and arcs, and it must contain at least one of each of these elements. Each place may contain one or more *tokens*, as suggested by place p1 in figure 7.5. When a transition *fires*, it removes exactly one token from each of its input places, and it inserts exactly one token into each of its output places. However, a transition may only fire if there is at least one token in each of its input places, i.e., if the transition is *enabled*. Whenever a transition fires, the position and number of tokens within the PN may change, taking the PN to a new *state*. Each state is also called a *marking*.

Petri nets may exhibit several properties, for example:

- A PN is said to be *bounded* if and only if, for each reachable marking, each place contains a finite number of tokens [David and Alla, 1992].
- A PN is said to be *safe* if and only if for each place the maximum number of tokens is one [David and Alla, 1992].
- A PN is said to be *live* if and only if for every reachable marking M' and every transition t, there is a state M" reachable from M' which enables t [Aalst, 1998].
- A PN is said to be *strongly connected* if and only if for every pair of nodes (places and transitions) x and y, there is a path leading from x to y [Aalst, 1998].

With places, transitions and arcs it is possible to define PNs with very different configurations. For the purpose of workflow modeling, however, only a subset of these possibilities is actually useful. A PN which represents a process definition is called a Workflow net (WF-net) [Aalst, 1998]. A PN is a WF-net if and only if it obeys two conditions. The first condition is that the PN must have a single input place and a single output place, as shown in figure 7.5. The second condition is that there should be no dangling places or transitions; every place and transition must have an active role in the PN. In other words, if the output place is connected to the input place via an extra transition t*, as shown in figure 7.5, the PN must be strongly connected .

But even if a PN satisfies these requirements, it is still possible that it contains inconsistencies, such as deadlock, livelock, or mismatch between branching and synchronization constructs. In figure 7.6(a), once t1 or t2 fire the token will go either place p1 or p2, respectively, so t3 and t4 will never be enabled. In figure 7.6(b) there is no deadlock, but t3 cannot be enabled so no token can reach the output place. In figure 7.6(c) there is a branching mismatch because both p1 and p2 will receive a token in parallel but then these tokens will be allowed to reach the output place independently. As a result, the output place will end up with two tokens for each token originally in the input place, which shows that the WF-net is not safe. Moreover, if the output place is connected to the input place via an extra transition t*, then the resulting PN is not bounded. Instead of having two separate transitions t2 and t3, there should be a single synchronizing transition for the two places p1 and p2.

If a WF-net has these or other kinds of inconsistencies, then the process may not behave correctly at run-time. To ensure that a process behaves correctly, the WF-net should satisfy two requirements. Assuming that in the beginning there is only one token at the input place, the first requirement is that, once a token reaches the output place, all other places should be empty. The second requirement is

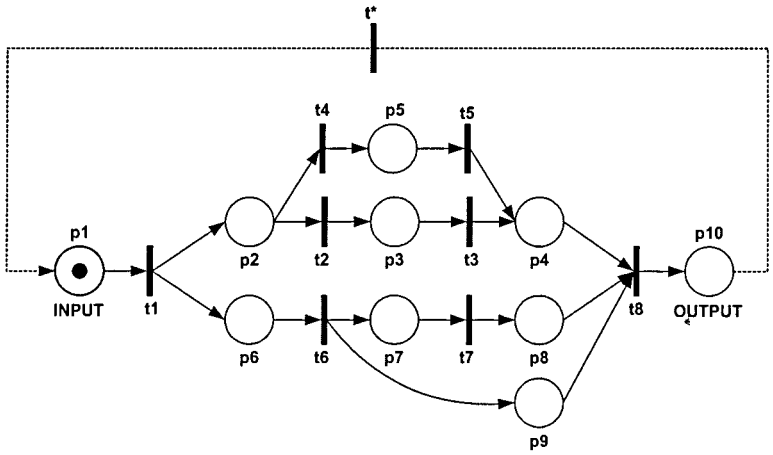


Figure 7.5: Example of a sound WF-net

that there should be no dead transitions: given all possible firing sequences, it should be possible to fire each transition at least once. These two requirements define the *soundness* property [Aalst, 1998]. An interesting result is that a WF-net is *sound* if and only if the PN obtained by connecting the output place to the input place, via an extra transition t*, is *live* and *bounded* [Aalst, 1998]. So by checking if a PN is live and bounded, it is possible to ascertain whether the WF-net is sound or not.

In section 6.3.2 on page 336 it was argued a workflow management system supporting the engineering of business networks should support process nesting. This means that a process can be defined as being composed of other sub-processes. If a process is described by means of a WF-net, then this implies that each node on the WF-net may refer to another WF-net, as suggested in figure 7.7. In order to ensure that the whole process behaves correctly, the resulting WF-net must be sound. However, in some cases there is no need to check the resulting WF-net for soundness. According to the compositionality theorem [Aalst, 2000b], if the containing WF-net is safe and sound and the contained WF-net is sound, then the resulting WF-net is sound. Furthermore, according to the same

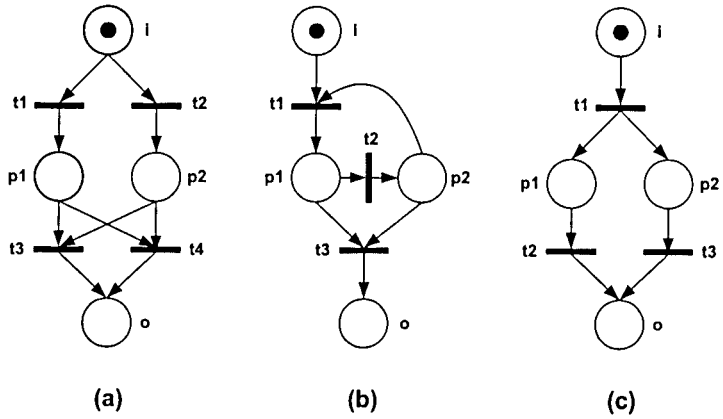


Figure 7.6: Deadlock (a), livelock (b) and branching mismatch (c) inconsistencies

theorem, if both the containing and the contained WF-nets are safe and sound, the resulting WF-net is also safe and sound.

7.1.2.2 State-based vs. event-based modeling

Originally, these results have been obtained by assuming that transitions represent activities and that places represent states between activities [Aalst, 1998]. This corresponds to a state-based view of process modeling, since the emphasis is put on states which determine which activities have been already performed, and which activities should be performed after a given state. However, Petri net theory assumes that transitions are instantaneous, whereas workflow activities may take an arbitrary amount of time to be performed. Thus, the state-based view is only appropriate if all activities have a relatively short duration. Activities with a relatively long duration should be modeled by a transition-place-transition module [Zhou and Venkatesh, 1999], in which the first transition represents the start, the place represents the activity being performed, and the second transition represents the end of the activity.

Applying this principle to all activities, the state-based view turns into an event-based view, where places represent activities and transitions represent the start or finish of activities. The presence of a token in a place means that the corresponding activity is being performed. When this activity completes, it generates an event which fires the output transition connected to this place. As the transition fires, it removes the token from that place and inserts the token into a new place, triggering the following activity. This event-based view is illustrated in figure 7.8, where each place is associated with an activity and each transition is associated with an event. Whenever a given activity completes, it generates an event that fires the output transition connected to its place. A transition without an associated event can be fired as soon as it becomes enabled.

Regarding this behavior, some special cases should be noted. The first special case is illustrated in figure 7.9(a), in which the place is connected to more than one output transition. In this case, there should a different event for each of these transitions: event E1 fires transition t1, while E2 fires t2. The same event cannot be associated with more than one transition, since this would result in a *conflict* [David and Alla, 1992]. Events E1 and E2 should be interpreted as alternative events, i.e., A1 produces only one of them, not both. This way it is possible for activity A1 to produce two outcomes with a different meaning. For example, if A1 is a budget approval activity, event E1 could mean that

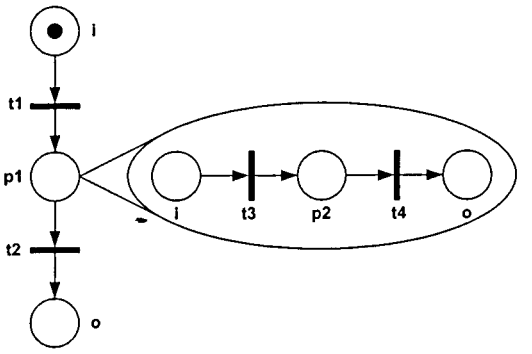


Figure 7.7: Compositionality of WF-nets

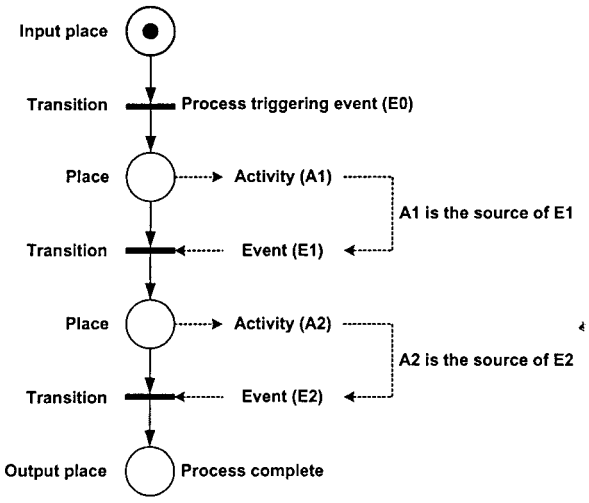


Figure 7.8: Event-based modeling

the budget was approved, whereas E2 could mean that the budget must be adjusted. This is effectively an event-based conditional transition as discussed in section 6.3.3 on page 337.

The second special case is illustrated in figure 7.9(b), where the output transition t3 has more than one input place. In this case, activity A1 generates event E1 but the attempt to fire the transition is unsuccessful because the transition is not enabled; only when activity A2 completes will it generate another event E1 which will fire the transition. However, if both activities are running at the same time, i.e. if both p1 and p2 contain a token, then a potential problem arises because the event that occurs first fires transition t3 before the second event is generated. This would mean that if activity A1 finishes first, for example, then the token will be removed from place p2 before activity A2 is actually complete. To avoid this situation, both p1 and p2 should have their own event-triggered transitions t1 and t2, which lead to two separate places with no associated activity and only then to a common transition t3 with no associated event, as shown in the shaded circle in figure 7.9(b). This way, the transition effectively records the occurrence of events so that t3 fires only after the last event has been issued.

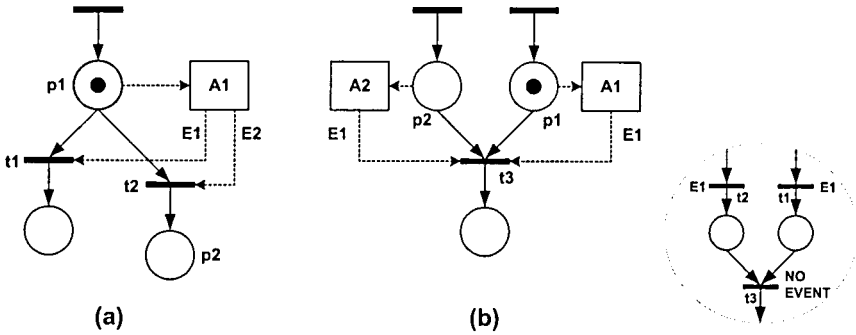


Figure 7.9: Special cases in event-based modeling

In this work, the use of event-based modeling is preferred rather than state-based modeling due to the following main reasons:

1. Transitions are supposed to be instantaneous, whereas tokens can remain in places for an arbitrary amount of time. Therefore, it seems more natural to associate activities with places and events with transitions. In fact, Petri net theory already allows events to be associated with transitions; this kind of Petri nets are called *synchronized Petri nets* [David and Alla, 1992].

2. In state-based modeling, the state of a Petri net represents a point in time when certain activities have been completed and other activities have not yet started. This state of idleness is very short-lived because as soon as one activity finishes other activities are immediately launched. If a human user is monitoring the execution of a given process, most of the time there will be some activity running. Thus, at a given instant, the process execution state should be defined as the set of activities that are currently running. This should correspond to the current marking in a WF-net.

3. With regard to composition of WF-nets, which corresponds to process nesting, it is more appropriate to enclose a sub-process in a place, as suggested in figure 7.7, rather than in a transition, as in [Aalst, 2000b]. This way, a sub-process has an input place and an output place just like any ordinary WF-net, rather than starting and ending with transitions. This approach is also more intuitive since the presence of a token in a place that refers to a sub-process means that the sub-process is running.

Formally, the two approaches - state-based modeling and event-based modeling - are equivalent, because it is possible to go from one representation to the other by means of simple substitution. Therefore, every model with activities associated with transitions can be expressed as an analogous model with activities associated with places. Figure 7.10 illustrates an example. In this case, each of the transitions "produce", "deliver", "remove" and "consume" in figure 7.10(a) have been replaced by a transition-place-transition module. Some of these new transitions, which delimit those activities, coincide with each other: for example, the transition "ready to deliver" in figure 7.10(b) is both the output transition for the "produce" place and the input transition for the "deliver" place. In addition, the place "production wait"/"consumption wait" was introduced in order to account for the fact that "produce"/"remove" may not begin immediately after "deliver"/"consume".

7.1.2.3 Colored Petri nets and process instances

Petri nets have been extended with several features such as events, time and color [David and Alla, 1992]. One of these extensions - events - is useful in event-based modeling. Another extension - color - is necessary in order to distinguish between process instances. Each token that is inserted in the input place of a WF-net represents a new process instance and, as such, it should have a unique color. Whenever a transition fires, it fires for a certain color, i.e., it removes exactly one token having that color from each of its input places, and it inserts one token with that color into each of its output places. For this reason, events must be colored as well, so that they make transitions fire only for a certain color. But if events are colored, then the activities that generate them must know which color to use. Hence, token color must be an input parameter for all workflow activities: they denote the process instance under which activities are performed.

Therefore, colored Petri nets provide an interesting mechanism to create and run several process instances using the same WF-net. This is clearly distinct from the way workflow management systems

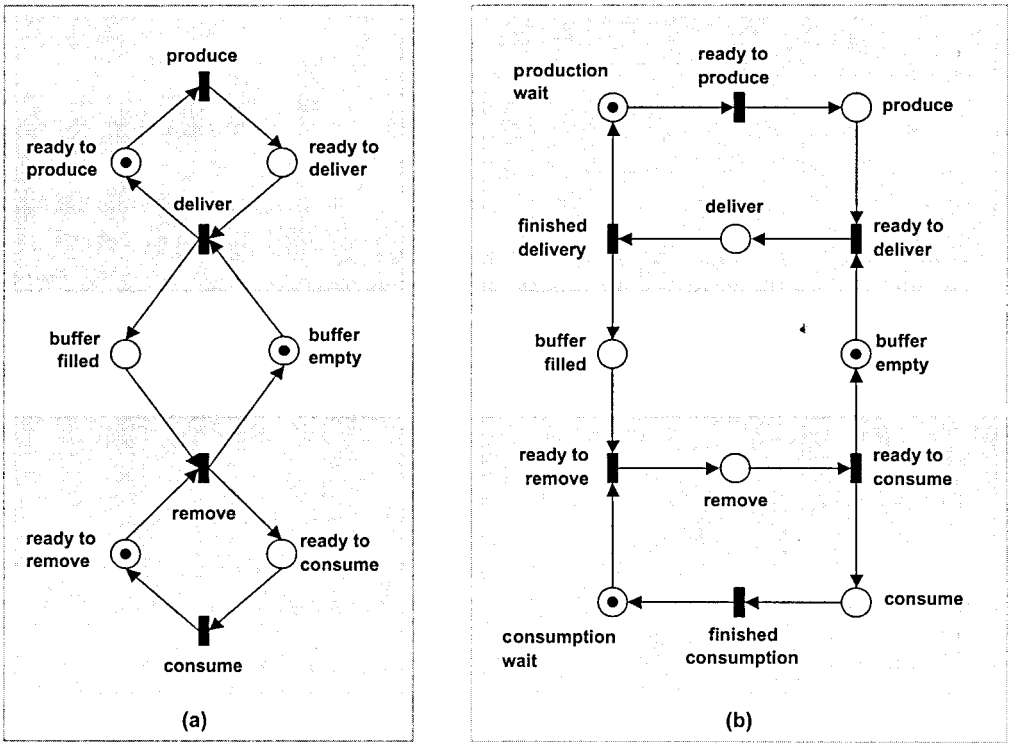


Figure 7.10: Equivalent representations of a producer-consumer system[129]

usually create and run process instances. Typically, workflow management systems create a new process instance by reading the original process definition and creating a run-time data structure that represents the process model to be executed. For example, in the DAMASCOS WfBB the WfFacility reads a WPDL process definition and creates a new WfProcess object containing several WfActivity objects, as explained in section 6.2.2.7 on page 324. The run-time data structure resembles the original process definition, but if that process definition is modified, the changes are not reflected onto the run-time data structure. In the case of colored WF-nets, the process definition is indeed the run-time data structure in which process execution is recorded as it unfolds. In fact, it is the run-time data structure for all process instances, since each new process instance is in effect just a different token color within the same WF-net.

7.1.2.4 Stochastic Petri net models

Each workflow activity takes a certain amount of time to be performed. In a manufacturing system, for example, the time it takes for a machine to perform certain operations may be fixed. But in an administrative department, on the other hand, the time it takes to perform some bureaucratic activity can only be given by an average value. The point is that it is not always appropriate to assume that activities have constant durations. In terms of WF-nets, as shown in figure 7.8, this means that the

[129]This example was inspired by the PN description available at http://www.informatik.hu-berlin.de/top/pnml/examples_prodcons.html

time it takes from the moment the token arrives at a place until it leaves this place may be different each time the activity runs.

This time is often referred to as the *transition delay*, and it is best modeled using a random variable. Usually, transitions delays are assumed to have an exponential distribution, although non-exponential functions have also been studied [Watson, 1991]. A Petri net in which transition delays are random variables is called a *stochastic Petri net* (SPN) [Balbo, 2001]. The main advantage of using exponential distributions is that it can be proved that a SPN is isomorphic to a Markov chain [Molloy, 1982]. Hence, the performance of a SPN, and thus the performance of a workflow process, can be analyzed using Markov chain techniques. The performance analysis of a SPN can be carried out as follows [Zhou and Venkatesh, 1999]:

1. The process is modeled as a PN with exponentially distributed time delays associated with transitions. To characterize these exponential distributions, it is enough to specify the average time delay for each transition. The inverse of this parameter is known as the transition's *firing rate*.

2. The second step is to generate the *reachability graph* for the PN, i.e., to identify all reachable markings, and all the transition firings that take the PN from one marking to the other. Each marking represents a particular state of the corresponding Markov chain, and the firing rates provide a measure of the transition probabilities between states in the Markov chain.

3. The Markov chain can then be analyzed using stochastic techniques. The goal is to compute the steady-state probabilities for the Markov chain, which can be obtained by solving a set of linear equations.

4. Given the transition firing rates and the steady-state probabilities for the Markov chain it is possible to compute several performance measures. The most common ones are: (a) the probability of reaching a particular marking or subset of markings, (b) the expected value of the number of tokens in a given place, and (c) the mean number of firings per unit time of a given transition.

Examples of this kind of performance analysis can be found in [Zhou and Venkatesh, 1999], [Marier et al., 1997] and [Ferscha, 1994]. There are also some tools that provide these analysis capabilities, such as SPNP[130] and GreatSPN[131]. Additional tools can be found in the Petri Nets Tool Database[132].

7.1.3 Interaction with resources

As discussed in section 6.3.1 on page 334, workflow management systems often assume particular ways in which they assign tasks to resources, and in which they invoke or interact with resources in order to fulfill those tasks. In practice, though, there are several possibilities of how the interaction with resources can take place. In DAMASCOS, the WfFacility interacts with WfBB users according to a simple request-reply mode, in which the reply message is supposed to bring back the task's output. In PRONEGI, there is an initial multicast request which users may accept or reject, and the first user to accept the task will be responsible for performing it; still, even after being accepted, the task can be rejected and the multicast request is sent again in order to assign the task to another user.

[130] http://www.ee.duke.edu/~kst/software_packages.html
[131] http://www.di.unito.it/~greatspn/
[132] http://www.daimi.au.dk/PetriNets/tools/db.html

At first sight, this issue might seem like a simple matter of having user assignment or role assignment, respectively. However, even within these task assignment strategies several kinds of interaction are possible. If an activity is assigned to a single user, this user may subsequently delegate the activity to another user, such as what happens in OfficeWorks (section 3.4.2.5 on page 100). If the activity is assigned to a role, the workflow enactment service may employ an activity assignment strategy such as the one used in PRONEGI, or it may keep track of the workload assigned to each resource so as to automatically choose the user with the smallest amount of work. These are just a few examples of the way in which the workflow enactment service may interact with resources; for a given scenario, this interaction may go from a simple request-reply mechanism to iterated attempts of task acceptance and fulfillment. In section 6.3.1 this was referred to as the resource allocation model.

The resource allocation model can be described as a workflow process, albeit at a lower level of granularity. In fact, the resource allocation model can be regarded as being a sub-process within the main process model, whose activities are to be assigned to resources, as suggested in figure 6.41 on page 336. The difference is that, whereas the process model deals with activities which are intended to produce some business-relevant result, the resource allocation model deals with smaller-size actions whose purpose is, for example, to send a message via the integration infrastructure.

7.1.3.1 Activities and actions

According to the event-based modeling approach discussed in section 7.1.2.2, each activity is initiated by a request and it produces one single event from a set of possibly several alternative output events. In other words, an activity begins with an input message and ends with an output message, which is perfectly in line with the fundamental character of a workflow activity, as discussed in section 6.3.5 on page 340. However, in some cases, such as during assignment or even during execution, an activity may generate intermediate events before producing the final output event. Usually, these events indicate that activity assignment or activity execution has advanced onto a new stage.

This behavior can be modeled by having several transitions and places for each single activity, as suggested in figure 7.11(a). The transitions are associated with the events that the activity generates, while the places represent execution stages that are delimited by those events. These event-delimited execution stages will be called *actions*. Since each transition is associated with one event, and given that two consecutive transitions in a Petri net effectively delimit one place, then it makes sense to associate places with actions, rather than to associate several places with a single activity. Figure 7.11(b) illustrates this modeling approach.

If an activity comprises just an input message and an output message, as in DAMASCOS, then a single action is enough to represent that activity. But if an activity requires several steps of interaction between the workflow enactment service and one or more resources, then the activity is best modeled as a sequence of actions. In PRONEGI, for example, there is a first action which sends a multicast request to all Subscribers under a given Context, and then a second action which confirms that the activity has been assigned to a certain user and removes the activity from the remaining task lists.

7.1.3.2 Resource invocation

Another issue is that a workflow management system may have to invoke several kinds of resources. Some tasks may be performed by human participants, whereas other tasks may have to be performed by manufacturing equipment, and still others may be performed as operations on a database, for example. Designing a workflow client application and requiring all resources to use that workflow

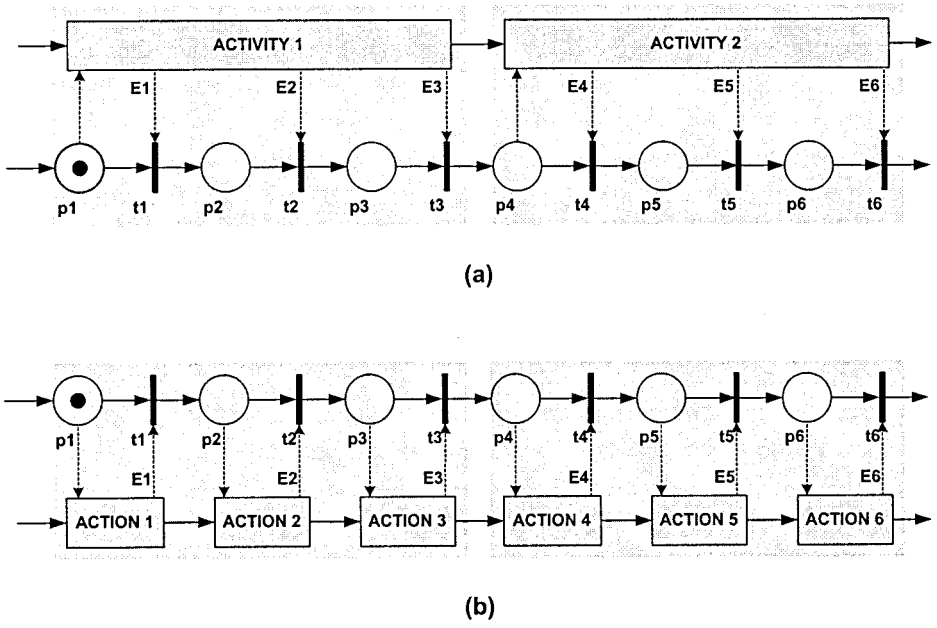


Figure 7.11: Dividing activities (a) into actions (b)

client application can be quite restrictive if resources other than humans must be invoked. For this reason, commercial workflow management systems such as Staffware or OfficeWorks allow activities to be performed by sending a new task to a workflow client application, or by invoking an external software program for which input and output parameters are provided.

A reusable workflow enactment service cannot assume that a certain workflow client application is available, or that external programs can be invoked in a certain way, because these details differ from one workflow management system to another. A workflow management system having a Web-based architecture, for example, may be able to deliver tasks to users via e-mail, and to invoke remote Web services; another workflow management system having a CORBA-based architecture may require the use of a particular workflow client application, and it may prefer to invoke remote CORBA objects. These details are specific to the workflow management system implementation, and to the resources available within a given scenario.

A reusable workflow enactment service, which is intended to provide process execution capabilities for any workflow management system in general, must refrain from implementing any mechanism concerning resource invocation. Everything the workflow enactment service should assume is that when a token arrives on a place a certain action must be invoked, regardless of what that action will effectively do. This action may dispatch a task to a workflow client application, it may request a remote machine operation, or it may perform some data operation on the local information system. Whatever it does, it is not the purpose of the workflow enactment service to implement these actions; instead, the workflow enactment service just invokes the action and then waits for it to return some event.

Hence, actions are more than just a modeling construct - they are also a wrapper API that provides access to any kind of resource. This wrapper API will be called the *resource interface layer*, as shown in figure 7.12. The actions in the resource interface layer are conceptually similar to the adaptors in

the wftk toolkit, as discussed in section 7.1.1; basically, they insulate the workflow enactment service from resource-specific details. On the other hand, actions also implement the observer pattern on behalf of those resources: the workflow enactment service invokes the action, and then it listens to events coming from that action, regardless of how the action actually invokes the resource.

In general, however, the action does not communicate directly with the resource. Instead, it relies on an integration infrastructure in order to interact with a remote application, which is the interface to the resource. If the resource is a human user then the interface is a workflow client application that lists all tasks assigned to that user; if the resource is a machine or computer program then the interface is a small wrapper application which translates the workflow task into one or more machine-specific commands. These client applications and these wrapper applications make up the *resource invocation layer*, as shown in figure 7.12.

Depending on how the resource invocation layer is implemented, and on how this layer interacts with resources, the observer pattern may be stretched until the ultimate resources or not. For example, the wrapper application for a certain machine may have to issue some commands first and then check if the commands have produced the desired result (pull model), as suggested in figure 7.13. This is not an observer behavior because there is no callback from the machine to its wrapper. But to the action, the machine may still appear to generate events because the wrapper application is able to notify the action that the commands have been executed successfully (push model). To the workflow enactment service, it always seems like the resources behave according to the push model.

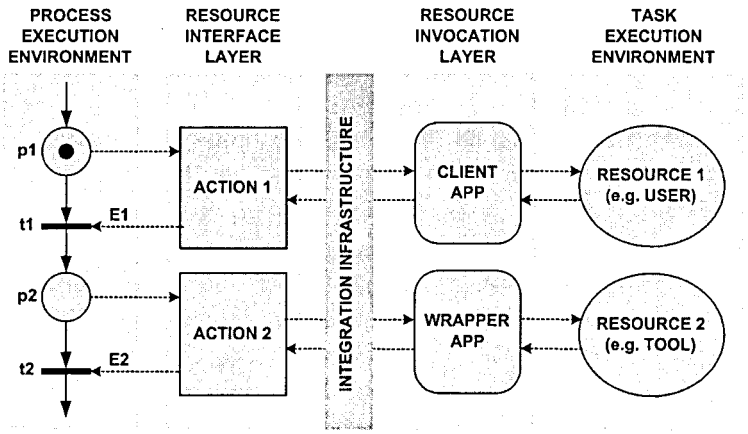


Figure 7.12: Actions as the resource invocation layer

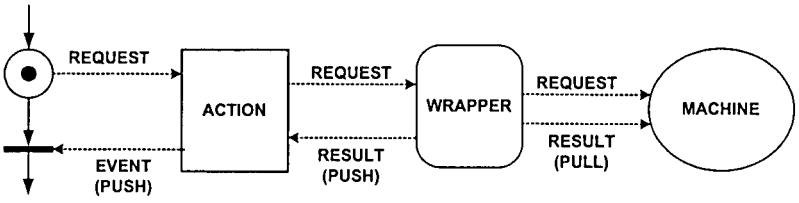


Figure 7.13: Push and pull models in resource invocation

7.1.3.3 Activity instances

Each single resource may be requested to perform actions that belong to several process instances. But each process instance is distinguished from other process instances by a unique token color, as explained in section 7.1.2.3. Therefore, the token color must be supplied as an input parameter to the resource, so that it distinguishes each requested action. On the other hand, when the action is completed it generates an event that triggers the transition of a token from the current place into the following place, as suggested in figure 7.13. This transition occurs for a certain process instance, hence for a certain token color. The generated event must specify this token color, so that the transition is fired for corresponding process instance only. Therefore, the token color is a parameter that should go all the way from the workflow enactment service to the resource, and then backwards from the resource to the workflow enactment service.

This requirement is somewhat similar to what happens in the PRONEGI system, where the functional operations are given a unique activity identifier as part of the invoking URL, as shown on page 302. This unique activity identifier must be returned back to the workflow enactment service when the functional operation completes its operation, so that the workflow enactment service can identify the activity that has just been completed. In the DAMASCOS WfFacility, the distinction between activities from different process instances is based on having a unique activity handler send the input XML document to the WfBB user. The WfBB user must to reply to that same activity handler, so the WfFacility always knows which activity instance has been completed.

7.1.3.4 Activity input and output data

Each workflow management system has its own way of conveying input data to resources, and of receiving output data from those resources. In PRONEGI, for example, the workflow enactment service exchanges name-value pairs with functional operations, as discussed in section 6.1.3.2 on page 299. In DAMASCOS, the WfFacility exchanges XML documents with WfBB users. A reusable, generic workflow enactment service should support the exchange of information items which may comprise properties (name-value pairs) and documents (document files). When an action is invoked, it is given a set of input properties and documents, and it is expected to produce another set of output properties and documents. Thus, actions consume information items by using them to invoke resources, whereas events are a source of information items by bringing back an action's result, as suggested in figure 7.14.

Tokens are the carriers of information items: they bring properties and documents to actions, and they take properties and documents from events. Additionally, whenever a transition is fired the properties and documents are collected from the existing tokens before these tokens are removed from their places, and those information items are subsequently loaded onto the new tokens that are inserted in the following places. Thus, the output data from one action will be the input data for the following actions. Much like an action, a process may also have input and output information items, as suggested in figure 7.14. When the first token is inserted into the WF-net, it is loaded with the process input data. When a token reaches the WF-net's output place, the properties and documents it carries represent the process output data.

7.1.4 The Workflow Kernel

The Workflow Kernel is a prototype of a reusable workflow enactment service. Basically, it is a core of workflow functionality that comprises several objects which can be created, manipulated

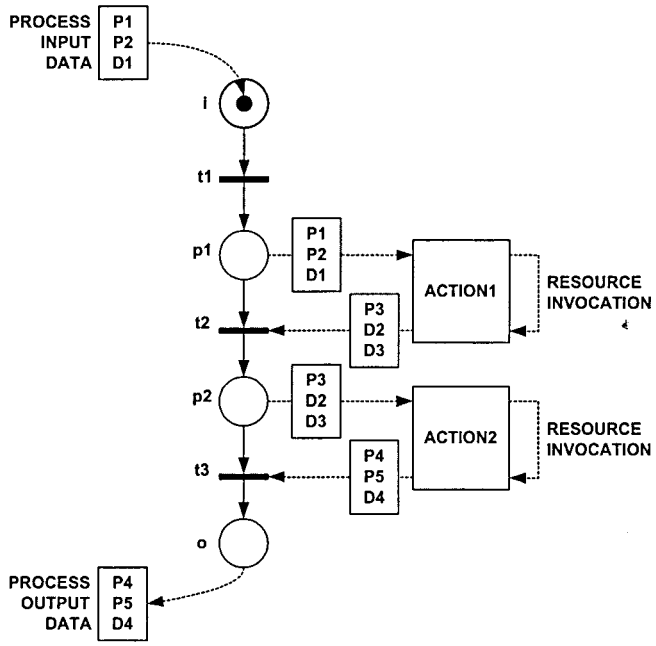


Figure 7.14: Input and output properties (P) and documents (D)

and augmented by external application code. These objects are interrelated according to the process modeling approach described in section 7.1.2, and to the resource invocation approach described in section 7.1.3. In general, there is one object representing each of the elements discussed in the previous sections, and each of these objects exposes its own interface. The relationships between these interfaces, which external applications have access to, effectively reflect the way the Workflow Kernel is internally implemented.

At the top of the class diagram, as depicted in figure 7.15, there is the Manager object which behaves as a singleton object, i.e., whenever applications attempt to create a Manager object, they will receive a reference to the same Manager object. This singleton Manager object is the point of entry for external applications, allowing these applications to retrieve the set of Process objects that exist in the Workflow Kernel. Since processes are represented by means of WF-nets, each Process object contains a set of places and transitions which are connected with each other by means of arcs. The arcs have not been explicitly represented as a separate object, but rather as a relationship between Place objects and Transition objects. Each Place object may be associated with an Action object, and it may contain an arbitrary number of Token objects. On the other hand, each Transition object may be associated with an Event object. Event objects are created by Action objects. Each Token, Action or Event object may contain an arbitrary number of Property and Document objects.

7.1.4.1 The IFeatures base interface

Each of the objects depicted in figure 7.15 implements its own interface. For example, the Manager object implements the IManager interface, the Process object implements the IProcess interface, the Place object implements the IPlace interface, the Property object implements the IProperty interface, and so on. Despite having different purposes, some of these objects have attributes in common. For

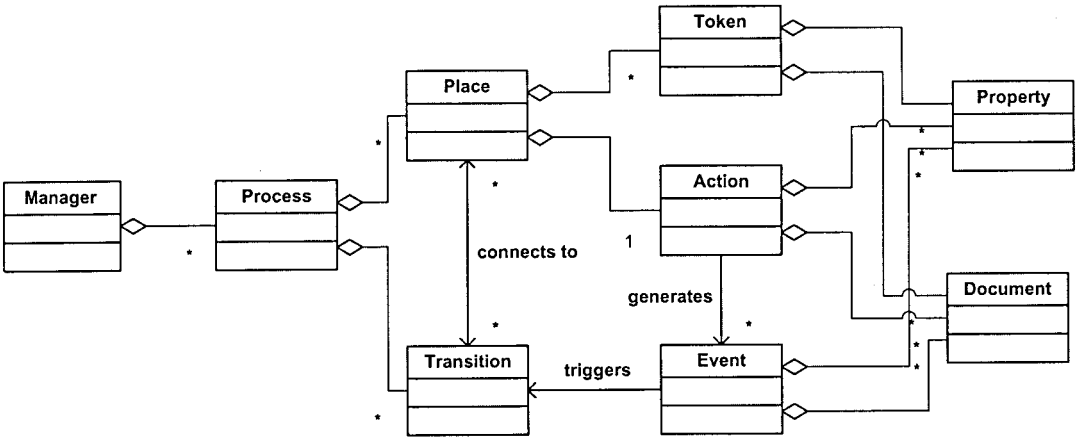


Figure 7.15: Simplified class diagram for the Workflow Kernel

example, most objects have a name attribute, some objects have a color attribute, and other objects such as Places and Transitions have graphical position coordinates. To avoid repeating attributes across objects and interfaces, it makes sense to encapsulate these common attributes in base interfaces, and to let each interface expose only those attributes which are unique to their corresponding objects.

But if a new base interface were to be defined for each set of common attributes, this would significantly increase the overall complexity of the Workflow Kernel interfaces. So, in order to simplify these interfaces but still have some sort of reusability, a single base interface was defined for all those objects, as shown in figure 7.16. This interface, called IFeatures, contains all the features that are common to at least two objects. This means that IFeatures may contain attributes that do not belong to all objects. For example, the color attribute (which is a string value) is only applicable to Tokens and Events, whereas the xpos and ypos attributes may only be applicable to Places and Transitions, if these are the only objects that can be represented graphically. Nevertheless, it was thought that it was better to provide objects with more attributes than those they require, rather than defining several base interfaces.

Most Workflow Kernel objects have a name attribute. In some cases, such as the Process, Place and Transition objects, the purpose of the name attribute is to provide a human-readable designation for those objects. In other cases, namely in the Document and Property objects, the name is a fundamental piece of information: a property must have a name in order to become a name-value pair, and a document requires a neutral designation other than the name of the file it represents. Besides the name, some objects also have a human-readable description. All Workflow Kernel objects have a unique identifier, which is necessary in order to serialize process objects. These objects are interrelated, e.g., places are connected to transitions. Since object references are volatile, the relationships between objects are stored as relationships between object identifiers.

Besides the identifier, name, description, color and position attributes, the IFeatures base interface contains a simulationmode attribute. This attribute is intended for Process and Action objects only, to allow them to run in a protected mode without interacting with resources, which is called the *simulation mode*. The main purpose of the simulation mode is to test process behavior prior to actual execution. In this mode, actions do basically nothing, but such a test is useful in order to ensure that

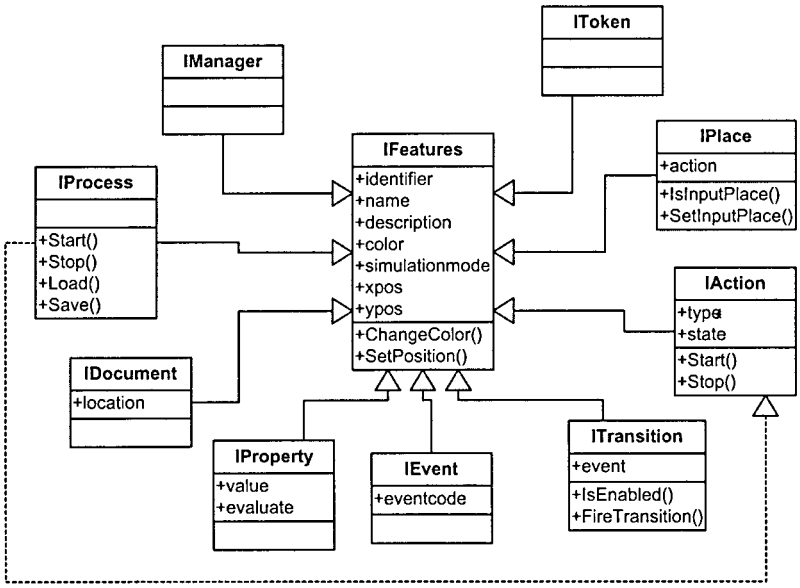


Figure 7.16: The IFeatures base interface and other Workflow Kernel interfaces

the process behaves as expected, and that the actions have been correctly configured. However, the simulation mode is just a user-friendly functionality; it is not intended to replace Petri net analysis. Incidentally, this must be performed with an external Petri net tool, since the current Workflow Kernel prototype does not provide analysis capabilities.

7.1.4.2 Object-specific interfaces

Each object-specific interface inherits all attributes and methods from the IFeatures base interface. Besides these attributes and methods, each object implements some attributes and methods of its own, as shown in figure 7.16.

IDocument The purpose of the IDocument interface is to provide access to Document objects, which have two main attributes: a name and a location. The name attribute is accessed from the IFeatures base interface, whereas the location attribute is accessed from the IDocument interface. The location attribute denotes the physical location of a document file which, in general, is located in the same machine as the Document object.

IProperty The IProperty interface provides access to Property objects and it is analogous to the IDocument interface, but instead of two it contains three main attributes: name, value and evaluate. The name attribute is accessed from the IFeatures base interface. The value attribute contains the property value and, ideally, it should be able to hold any kind of data type. In CORBA, for example, this attribute should be of type any; in COM it would be of type VARIANT. The evaluate attribute contains the name of another property whose value should become also the value for this property, i.e., the evaluate attribute allows the property value to be assigned the value of another property.

IEvent The Event object contains two main attributes: color and eventcode. The color attribute is a string value accessible from the IFeatures base interface. The eventcode attribute is a numerical value that distinguishes an Event object from other Event objects produced by the same Action. In general, the eventcode attribute has some semantic meaning, and it may be used to implement branching decisions within a Process, according to what was discussed in section 6.3.3 on page 337. Each branching alternative can be represented by a Transition with an associated Event which has a different eventcode from the remaining alternatives.

ITransition The Transition object has an event attribute which may contain a reference to an Event object or not. If the event attribute contains a valid reference to an Event object, then the Transition is said to be associated with that Event. A Transition that is associated with an Event E1 will fire for a received Event E2 if (a) Event E2 has the same eventcode as E1, and if (b) the Transition is enabled for the color of E2. If the event attribute is null then the Transition is associated with no Event, so it will fire as soon as it becomes enabled for some color, i.e., as soon as each of its input Places contain at least one token with the same color, regardless of which color it actually is.

The Transition object contains two methods: one that tells whether the Transition is enabled for a given color, and another that attempts to fire the transition. The FireTransition() method accepts an Event as input parameter, but it also allows this parameter to be null. If the supplied Event is null, then FireTransition() will attempt to fire the Transition as if no Event were associated with it, i.e., it will fire the Transition for all colors that it is enabled. The attempt to fire the transition occurs regardless of whether the Transition has an associated Event or not. If a valid Event is supplied as input parameter, then FireTransition() will attempt to fire the Transition for the color of that Event, regardless of the eventcode attribute.

IPlace The Place object has an action attribute which may contain a reference to an Action object or not. If the action attribute contains is a valid reference to an Action object, then the Place is said to be associated with that Action. The Action will be started as soon as a Token arrives at the Place, and it will be stopped when the that Token is removed from the Place, as the result of a Transition having fired. If the action attribute is null then the Place is associated with no Action, so nothing happens when a Token enters or leaves the Place.

A Place object can represent a regular place or a *process input place*. A process input place is a place that receives a token when the process is started, i.e., it is the WF-net's input place. The SetInputPlace() takes a boolean parameter as input, which specifies whether the Place should be a process input place (true) or not (false). The IsInputPlace() method returns a boolean value with the same meaning. By default, Place objects are assumed to represent regular places.

IAction The Action object contains two main attributes: type and state. The type attribute distinguishes the purpose of an Action from other kinds of Actions. For example, an Action A1 that sends a task to a worklist has a different type from an Action A2 that invokes a remote application program. The state attribute stores a numerical value that denotes the Action's execution status, i.e., whether it is inactive, running or complete. The Start() method initiates the action, and it is invoked when a Token enters the Place which the Action is associated with. The Stop() method releases any computational resources that were required during the Action's operation, and it is invoked when a Token leaves the Place. Both methods take the Token's color as an input parameter.

IProcess The Process object contains methods that allow the process to be run, and to be saved or loaded from a file. The Start() method initiates a new process instance by inserting a Token into each Place that is a process input place. The Stop() method removes all Tokens with a given color from every Place, therefore discarding the corresponding process instance; stopping a process instance by calling this method is an irreversible command (i.e., there is currently no pause/resume capabilities). The Save() and Load() methods save and load the Process and all of its constituent elements to or from an XML document file. The Workflow Kernel automatically saves Processes when it shuts down, and it loads them when starting up.

In order to support process nesting, the Workflow Kernel allows a Process to be run as a sub-process inside another Process. A sub-process can be regarded as being just a special kind of Action which, like other Actions, must be associated with a Place. But since all Actions must implement the IAction interface, then Process objects must also implement this interface, as suggested in figure 7.16 by the inheritance relationship between IAction and IProcess. In this case, the Action's Start() and Stop() methods correspond to the Process's Start() and Stop() methods.

7.1.4.3 The IEnumerator base interface

Most Workflow Kernel objects contain other objects, as shown in figure 7.15. For example, each Process contains a set of Places, each Place contains a set of Tokens, and each Token contains a set of Properties and a set of Documents. This means that each object may have to maintain one or more collections of other objects. In practice, these will be collections of object references rather than collections of objects, because each object is created separately. Each of these collections must allow the container object to add, remove and retrieve object references that are stored in that collection. For example, the Process object is a container of Place and Transition references, and a Token object is a container of Property and Document references.

Since most Workflow Kernel objects require the ability to maintain a collection of object references, it would make little sense to specify object-specific methods to deal with collections. Instead, just like IFeatures provides attributes that are common to all objects, there is a base interface that provides methods that are common to all containers, and that allow a container to add, remove and retrieve references from an object collection. This base interface is called IEnumerator, and it is depicted in figure 7.17.

The IEnumerator interface is the base interface for all collection containers which, incidentally, do not have any attributes or methods beyond those provided by IEnumerator. Thus, the methods that give access to a collection are the same for all containers. But each container deals with a different type of objects; for example, a Place container has a collection of Place references, and a Transition container has a collection of Transition references. If IEnumerator's methods would be designed to deal with a collection of Places, they would not be suitable to deal with a collection of Transitions. Therefore, the use of IEnumerator is only possible if all objects share a common base interface, so that IEnumerator's methods refer to this base interface rather than referring to object-specific interfaces.

In fact, Workflow Kernel objects share a common base interface - IFeatures - so IEnumerator's methods should take IFeatures pointers as input and output parameters. In other words, IEnumerator and all other container interfaces provide access to a collection of references to objects which implement the IFeatures interface. For IPlaceContainer, for example, these objects are supposed to be Places; for ITransitionContainer, the objects are supposed to be Transitions. If an application obtains an IEnumerator pointer, then it can use IEnumerator's type attribute to find out which type of objects

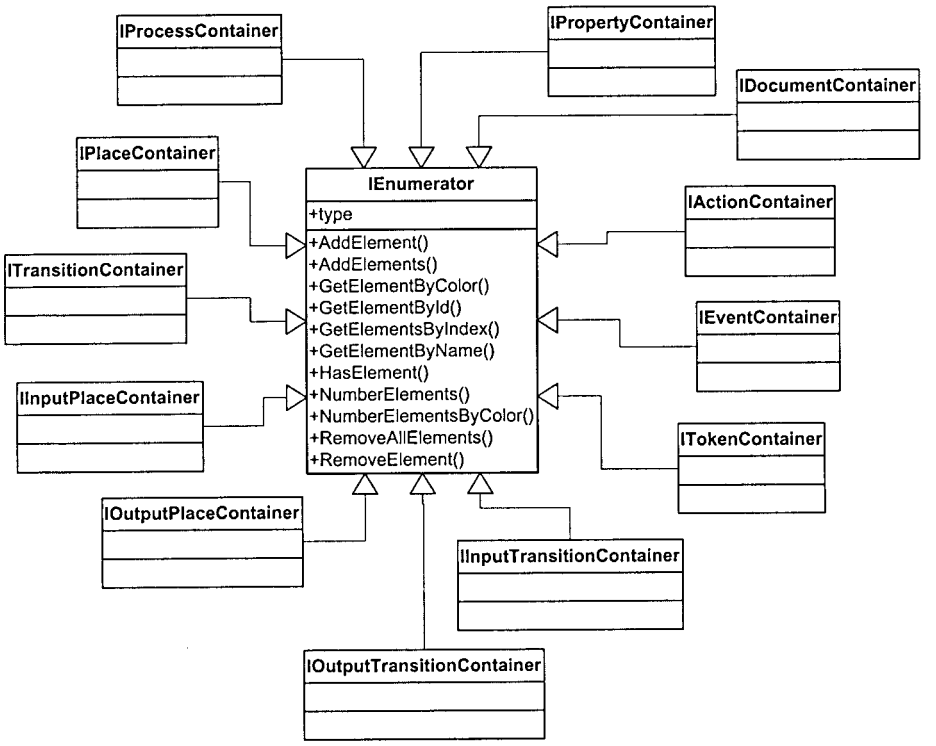


Figure 7.17: The IEnumerator base interface and other container interfaces

the collection holds. The type attribute is a numeric value that determines whether the IEnumerator pointer actually represents an IProcessContainer, an IPlaceContainer, an ITransitionContainer, etc.

Regarding IEnumerator's methods, the AddElement() method adds a single IFeatures pointer to the collection, whereas AddElements() adds the contents of another IEnumerator. The AddElements() method only succeeds if both IEnumerators have the same type value; otherwise, it will return an error. The GetElementByColor(), GetElementById() and GetElementByName() methods return an IFeatures pointer to the first object that matches the color, identifier or name being supplied. The HasElement() method tests whether an given reference already belongs to the collection. The NumberElements() method returns the total number of references in the collection, and NumberElementsByColor() returns the number of different colors within the collection. RemoveElement() removes a single reference, and RemoveAllElements() empties the collection.

Some Workflow Kernel objects implement more than one container interface. The Manager object is a Process container so it implements IProcessContainer; in addition, it implements IActionContainer in order to maintain a list of all the available types of Actions in the system. The Process object is a container of Places and Transitions, so it must implement both IPlaceContainer and ITransitionContainer; and it also implements IEventContainer in order to maintain a list of all Events produced by Actions within that Process. A Place may have input and output Transitions, so it implements both IInputTransitionContainer and IOutputTransitionContainer; a similar situation occurs with Transition objects, which implement both IInputPlaceContainer and IOutputPlaceContainer. The container interfaces that each object implements are shown in figure 7.18.

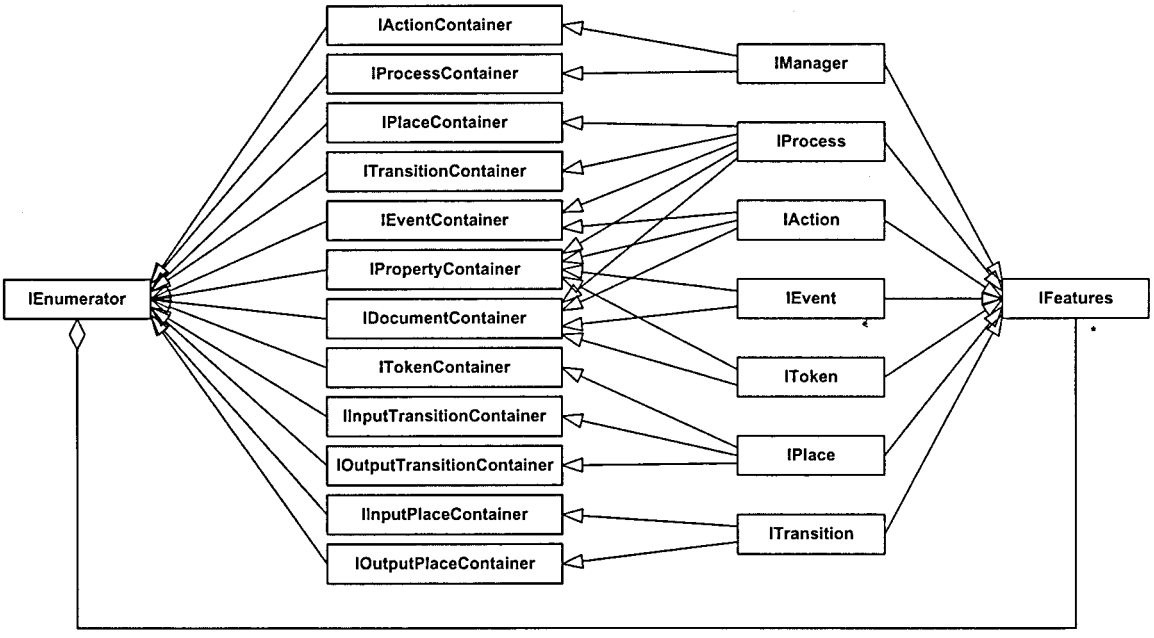


Figure 7.18: Relationship between container and object-specific interfaces

7.1.4.4 The INotifySink callback interface

As shown in figure 7.11(b), every Action is expected to produce an Event. This Event represents that Action's result: it has a certain eventcode value, a certain color, and it carries information items, namely Properties and Documents, which are the Action's output. When this Event arrives back at the Process it will trigger those Transitions which (1) have the same eventcode value associated with it, and which (2) are enabled for the Event's color. The Event is delivered to or pushed onto the Process, as suggested in figure 7.13. Since the Action takes the initiative to produce and deliver the Event, the Process must provide some interface that the Action can invoke in order to push back the Event - this interface is called INotifySink.

The INotifySink interface is a callback interface because it was specified for one object (Action) but it is implemented by another object (Process). In other words, the INotifySink interface specifies how an Action can deliver Events to other objects, provided that these objects implement INotifySink. This interface has a single method called OnNotify(), which takes an Event as input parameter. The main purpose of INotifySink is illustrated in figure 7.19: whenever a Token arrives at a Place, the Action associated with that place is started and, while the Action is running, it produces an Event that is delivered back to the Process via INotifySink. In this case, a Process is an observer of Events produced by Actions.

In this scenario, the INotifySink interface provides an event notification mechanism within the Workflow Kernel. But the purpose of Workflow Kernel is to provide a reusable workflow enactment service, so it has to be combined with additional components in order to create a complete workflow management system. Therefore, whatever event notification mechanisms are employed within the Workflow Kernel, these event notification mechanisms may have to be extended to all components within a workflow management system. This requirement becomes quite challenging if one realizes

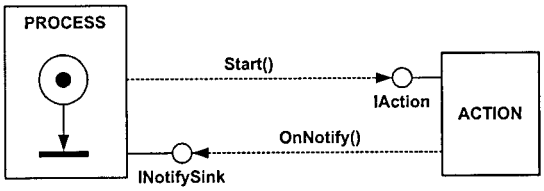


Figure 7.19: Internal event notification with INotifySink

that, for a reusable workflow enactment service, most the components it interacts with are not known beforehand.

To understand this requirement one can refer to the MENTOR architecture, for example, which is depicted in figure 3.24 on page 108. The MENTOR architecture proposes a distributed workflow management system where several workflow engines can collaborate in executing processes described by state charts. The system is inherently transactional by employing an underlying CORBA-based transaction-processing monitor (TP monitor). A typical transaction, run by a single workflow engine, includes sending messages to work-lists or to other workflow engines, and logging those actions in a workflow database. However, all transactions - and therefore all resource invocation possibilities - have been hard-coded inside the workflow engine [Wodtke, 1997]. Several components - for example the Log Manager, the Worklist Manager and the History Manager - are already provided, but it is not possible to augment the system with further functionality. Eventually, MENTOR would be superseded by MENTOR-lite, which clearly distinguishes between functionality that is intrinsic to the workflow engine and external components that can be integrated with that workflow engine. By making this distinction, MENTOR-lite has resulted in an extendable workflow engine [Weissenfels et al., 1998].

Getting back to the Workflow Kernel, one realizes that a reusable workflow enactment service must be able to notify external components of events that occur within that workflow enactment service[133]. This feature is essential in order to allow legacy systems or even future components to react to workflow events, bringing in specific functionality that the workflow enactment service itself does not provide (because it is not its purpose to provide). For example in the MENTOR architecture the Log Manager, the Worklist Manager and the History Manager should be regarded as external components that are observers of workflow events. Therefore, components that are external to the Workflow Kernel but still require to be notified of workflow events must provide an interface that allows the Workflow Kernel to notify them.

This interface must be specified as a callback interface because the Workflow Kernel knows nothing about the external components it will be integrated with, so it does not know which specific interfaces these components implement. Therefore, the Workflow Kernel must be the first to specify the callback interface that external components should implement in order to receive events. A further simplifying assumption, that is introduced in order to avoid defining more than one event notification interface, is that this callback interface should be same interface that Actions use to communicate Events to Processes, i.e., it should be INotifySink, as suggested in figure 7.20.

[133]In this context, "events" are not restricted to an Action-generated Events; it refers to any kind of relevant occurrence within the workflow enactment service.

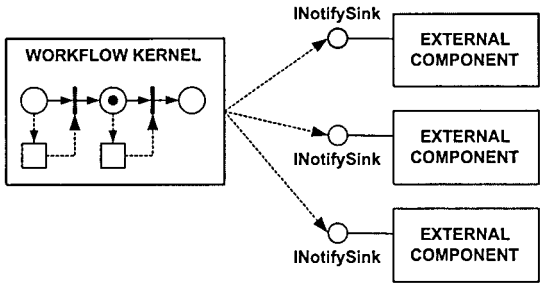


Figure 7.20: External event notification with INotifySink

7.1.4.5 The event notification method

A second issue concerning event notification is what kind of workflow events would external components be interested in receiving. Up to this point, only Events produced by Actions are notified. This could be enough for a logging application which keeps track of all Actions' results; but a monitoring tool, for example, could require the Workflow Kernel to notify occurrences such as Actions being started, and Tokens being added or removed from places; and a process editing tool could require the Workflow Kernel to notify other occurrences such as when new Places and Transitions are added to the WF-net, or when new Actions and Events are associated with Places and Transitions. In essence, and to be as generic as possible, the Workflow Kernel should be able to notify any change, in any of its objects, to any object that implements INotifySink.

There are two important consequences arising from this requirement. The first consequence is that every object within the Workflow Kernel must produce a notification event whenever one of its attributes changes or whenever one of its object collections changes. For example, a Property will produce a notification event if its name, value or evaluate attribute changes; a Place will produce a notification event when it is given a new Token, or when its input or output Transitions change. In general, both the object-specific interfaces and the container interfaces define methods that allow an object to be changed in some way; each of these changes will produce a notification event.

The second important consequence is that INotifySink's OnNotify() method must be able to convey precisely what has changed in which Workflow Kernel object. For this reason, the OnNotify() method has the following five input parameters:

```
OnNotify(process, obj_type, interface, ntf_type, any)
```

- The first input parameter is a pointer to the IProcess interface for the Process object within which the change has occurred.
- The second input parameter is an enumeration value (obj_type) that identifies the type of interface supplied as the third input parameter. This value identifies one of the specific interfaces discussed in section 7.1.4.2 or one of the container interfaces discussed in section 7.1.4.3. For example, the enumeration value TYP_FEATURES represents the IFeatures interface, whereas ENM_PLACE represents IPlaceContainer. The complete list of possible values for this parameter is shown in figure 7.21(a).
- The third input parameter is a pointer to the interface which was used to change the object. This parameter is supplied as a base interface pointer which must be subsequently converted

to a specific interface pointer according to the previous parameter value, as suggested in figure 7.21(a).

- The fourth input parameter is an enumeration value (ntf_type) that identifies the type of change. For example, NTF_NAME implies a change in an object's name, and NTF_ADDELEMENT means that a new object has been added to a collection. The complete list of possible values for this parameter is shown in figure 7.21(b). Naturally, the type of change that is specified in this parameter depends on the interface that was invoked; for example, the NTF_NAME value is possible if and only if the second input parameter specifies TYP_FEATURES, and the NTF_ADDELEMENT value is possible if and only if the second parameter specifies a container interface. The relationship between an interface and the notification events produced by invoking methods of that interface is illustrated in figure 7.22. The NTF_ACTIONEVENT value is used to convey an Action-generated Event object.

- The fifth input parameter contains a new value for the attribute that was changed, or a reference to the object that has just been added or removed from a collection. Basically, the purpose of this parameter is to contain any data that is relevant to the change that was done. For example, if the fourth parameter contains the value NTF_NAME then this parameter contains the new name that was given to the object; if the fourth parameter is NTF_ADDELEMENT then this parameter contains a reference to the object that has just been added to the collection, as shown in figure 7.21(b). If implemented with CORBA, this parameter should be of type any; in COM, it would be of type VARIANT.

7.1.4.6 Event sources and event sinks

Any object that implements INotifySink is able to receive events produced within the Workflow Kernel. An object that produces events will be called an *event source*, and an object that receives events will be called an *event sink*. An event source is any object that invokes INotifySink, whereas an any object that implements this interface can be an event sink. However, the fact that an object implements INotifySink does not necessarily imply that it will start receiving notification events. First, the object must subscribe to events produced by the event source, and only then will it start receiving events from that event source. When it is no longer interested in receiving events, the event sink must unsubscribe.

The methods that allow an event sink to subscribe and unsubscribe to events from an event source have not been specified during Workflow Kernel design; as will be shown ahead, they are provided by the chosen implementation technology. Anyway, all Workflow Kernel objects are event sources since they must be able to notify other objects of changes in their attributes and collections. Hence, all objects must implement the methods that allow an event sink to subscribe to events from an event source.

For example, if an event sink is interested in being notified whenever a Token enters or leaves a Place, then the event sink must subscribe to the events produced by that Place. If the event sink does so, it will also be notified of all other changes occurring in that Place, such as changes in its name or associated Action. Still, if the event sink is only interested in changes concerning Token insertion or removal, then it will only pay attention to OnNotify() calls in which (1) the second parameter is ENM_TOKEN, and (2) the fourth parameter is either NTF_ADDELEMENT or NTF_REMOVEELEMENT.

obj_type Value	3rd parameter
TYP_FEATURE	IFeatures
TYP_ACTION	IAction
TYP_DOCUMENT	IDocument
TYP_EVENT	IEvent
TYP_PLACE	IPlace
TYP_TRANSITION	ITransition
TYP_PROCESS	IProcess
TYP_PROPERTY	IProperty
TYP_TOKEN	IToken
TYP_MANAGER	IManager
ENM_ENUMERATOR	IEnumerator
ENM_ACTION	IActionContainer
ENM_DOCUMENT	IDocumentContainer
ENM_EVENT	IEventContainer
ENM_PLACE	IPlaceContainer
ENM_TRANSITION	ITransitionContainer
ENM_PROCESS	IProcessContainer
ENM_PROPERTY	IPropertyContainer
ENM_TOKEN	ITokenContainer
ENM_INPUTPLACE	IInputPlaceContainer
ENM_OUTPUTPLACE	IOutputPlaceContainer
ENM_INPUTTRANSITION	IInputTransitionContainer
ENM_OUTPUTTRANSITION	IOutputTransitionContainer

(a)

ntf_type Value	5th parameter
NTF_COLOR	new color (string)
NTF_XPOS	new xpos (string)
NTF_YPOS	new ypos (string)
NTF_NAME	new name (string)
NTF_DESCRIPTION	new description (string)
NTF_SIMULATIONMODE	new mode (numerical)
NTF_ACTIONTYPE	new type (numerical)
NTF_ACTIONSTART	color (string)
NTF_ACTIONSTOP	color (string)
NTF_ACTIONEVENT	Event (IEvent)
NTF_LOCATION	new location (string)
NTF_EVENTCODE	new code (numerical)
NTF_ACTION	new Action (IAction)
NTF_INPUTPLACE	(boolean)
NTF_EVENT	new Event (IEvent)
NTF_PROCESSCOMPLETE	Event (IEvent)
NTF_VALUE	new value (any)
NTF_EVALUATE	new evaluate (string)
NTF_ADDELEMENT	new element (IFeatures)
NTF_REMOVEELEMENT	old element (IFeatures)

(b)

Figure 7.21: The obj_type (a) and ntf_type (b) enumeration values

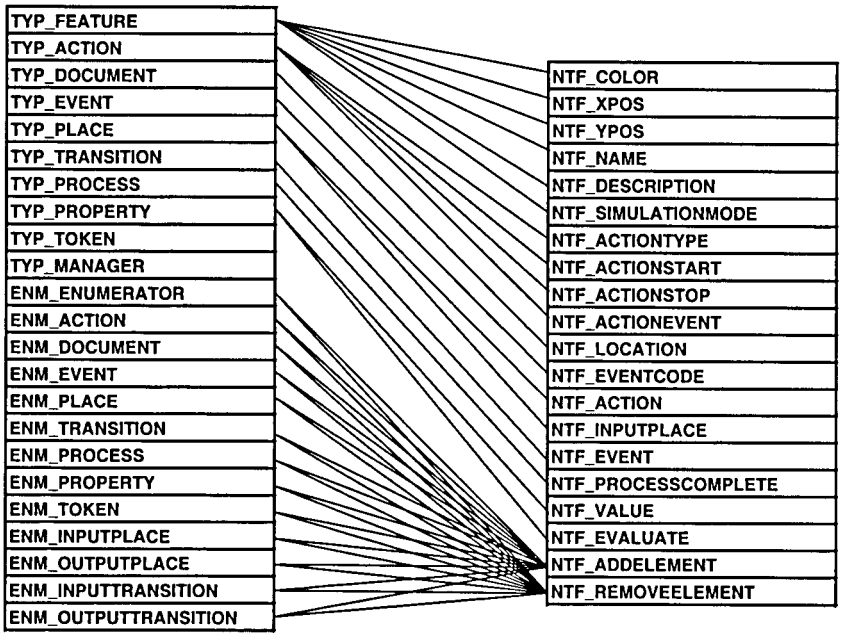


Figure 7.22: Relationship between obj_type and ntf_type values

However, this event notification approach has a major drawback: since all Workflow Kernel objects are event sources, an event sink may have to subscribe to several event sources in order to receive all the events it is interested in. For example, a monitoring tool that keeps track of process execution would perhaps be interested in receiving all events produced within a given Process, so it would have to subscribe to all event sources from the Process down to individual Properties and Documents. This is not only impractical but also difficult to implement: if new elements, such as Places or Transitions, are added to the Process then the monitoring tool would have to subscribe to events produced by these new elements.

If there is an event sink that is interested in receiving all events from a given Process, it makes more sense to allow this event sink to subscribe only once to events produced within a single Process, regardless of how many elements that Process may contain. In the same line of thinking it seems to be appropriate, for example, that if an event sink subscribes to events produced by an Action, it should also receive notification events from all Properties and Documents that the Action contains. In general, if an event sink subscribes to events from a certain object, then it should also receive events from all event sources which that object contains. For example, the highest amount of notification events will be received by subscribers of the top-level Manager object, which is (directly or indirectly) the container for all objects within the Workflow Kernel, as suggested in figure 7.15 on page 367.

To enable this kind of behavior, it was established as a mandatory requirement for the Workflow Kernel that all collection containers must automatically subscribe to events produced by the objects they contain. This way, if an external event sink subscribes to an event source, it is effectively subscribing to the events produced by that event source and by all other objects that event source contains. In addition, in some associations between objects, notably the association between a Place and an Action, an object subscribes to events produced by the object it is associated with. In general, any object that acquires a reference to another object will subscribe to events produced by this object.

This requirement implies that events will flow across the Workflow Kernel, down from Properties and Documents up to the singleton Manager object, as suggested in figure 7.23. This event flow mechanism makes of each Workflow Kernel object both an event source and an event sink. The object is an event sink for the events produced by objects it contains, and it is an event source for the events it produces by itself and for the events produced by the objects it contains.

The Workflow Kernel can take advantage of this event notification mechanism by using it to implement process execution behavior. Figure 7.24 illustrates an example of an event notification sequence that takes place during process execution. In this example, an Action A1 has been invoked and it is now returning an Event object. The Action notifies this event to Place P1 which is associated with A1 (step 1). For this purpose, A1 invokes P1's OnNotify() method:

```
P1.OnNotify(null, TYP_ACTION, IAction, NTF_ACTIONEVENT, IEvent)
```

where the first parameter, an IProcess pointer, remains unspecified (it will be filled in later on as the notification sequence progresses). Optionally, Action A1 may also notify the same event to an external event sink that has explicitly subscribed to this Action's events. Place P1 will forward this event to the Process object, which is that Place's container (step 2). For this purpose, P1 makes a similar method call as A1 did, but now invoking the Process's OnNotify() method; for the time being, the first parameter still remains unspecified:

```
Process.OnNotify(null, TYP_ACTION, IAction, NTF_ACTIONEVENT, IEvent)
```

Optionally, the Place may notify event sinks other than the Process. Once it is notified, the first thing the Process object does is to forward this event to its subscribers (step 3), which include the

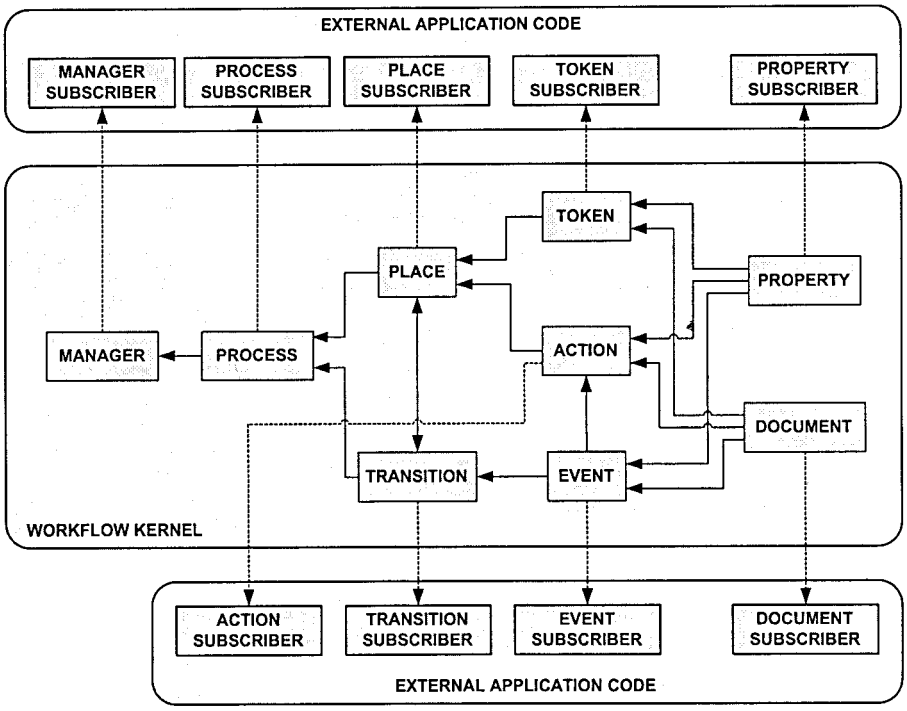


Figure 7.23: Flow of notification events across the Workflow Kernel

singleton Manager object and any external event sinks. Hence, the Process object invokes those objects' OnNotify() method, passing its own interface pointer as the first parameter:

Manager.OnNotify(IProcess, TYP_ACTION, IAction, NTF_ACTIONEVENT, IEvent)

After this chain of events is completed, the Place walks through all of its output Transition objects in order to find out which Transitions should be fired for that Event. In this example, the Place object finds that it should fire Transition T1, so it invokes T1's FireTransition() method, passing the received IEvent pointer as input parameter (step 4). When the Transition fires it removes Tokens from its input places (not shown in figure 7.24) and inserts Tokens into its output places (step 5). When the output place P2 receives a new Token, it starts its associated Action (step 6). It should be noted that steps 5 and 6 will result in new events being generated (Token insertion and Action started, respectively). Each of these events will follow the same notification sequence as described for steps 1, 2 and 3, but now OnNotify() will be filled in with the following parameters:

OnNotify(null/IProcess, ENM_TOKEN, ITokenContainer, NTF_ADDELEMENT, IToken)

OnNotify(null/IProcess, TYP_ACTION, IAction, NTF_ACTIONSTART, color)

7.1.4.7 Handling properties and documents

According to figure 7.18, there are four classes of Workflow Kernel objects which are Property and Document containers: Process, Token, Action and Event. A Process is a Property and Document container because, as suggested in figure 7.14, a Process has input and output information items.

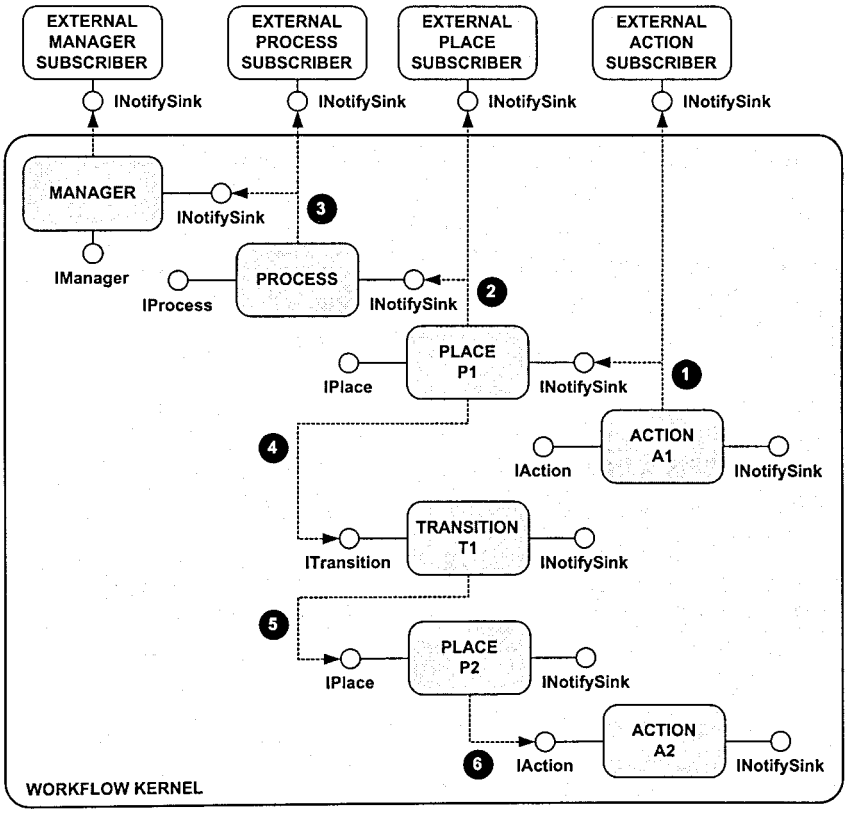


Figure 7.24: Process execution as a reaction to event notification

In practice, though, only input Properties and Documents are kept in the Process's Property and Document collections, respectively. The output Properties and Documents will be held by the Tokens that reach the Process's output Place, where each Token color represents a different workflow process instance. These output information items are basically the same as the input information items but now they may have different attribute values, and the set of output information items may contain many more elements than the original set of input information items. These new values and elements come from Actions' outputs during process execution.

The Process's input Properties and Documents should be configured when the Process is being defined, and before the Process is started. The input Properties represent configuration parameters for some or all of the Process's Actions, whereas the input Documents contain input data that will be provided to those Actions. These information items will be provided to Actions by means of Token flow across the Process's Places. The Actions' Properties and Documents should also be specified when the Process is being defined. Each Action's Property may be given a pre-defined value, or it may be assigned the value of another Property via the evaluate attribute. Each Action's Document may be given a pre-defined location or not; if the Document has no pre-defined location then either it is assumed that it will be assigned the location of a Process's Document or it is supposed to be an output Document that will be created only when the Action is performed.

When the Process is started, a new Token with a new color is inserted into the Place which is flagged as being a process input place. This Token will be loaded with the Process's input Properties

and Documents. If the Place has no Action associated with it then the Token will just wait for a following Transition to fire. If the input Place has an Action associated with it, then the Process will do two things: (1) transfer the Token's Properties and Documents to the Action's Properties and Documents, and (2) start the Action. The second step is just a matter of invoking the Action's Start() method passing the Token's color as input parameter. The first step is attained by doing the following:

- For each Property in the Token, the Process object checks if there is any Action's Property whose evaluate attribute contains the name of the Token's Property. If so, the Process assigns the value of the Token's Property to the Action's Property.

- For each Document in the Token, the Process object checks if there is any Action's Document with the same name. If so, the Process assigns the location of the Token's Document to the Action's Document. If the Token has several Documents with the name and this name matches the name of an Action's Document, then all of those Token's Documents are added to the Action as well.

If the Place is not the Process's output place, then it will have one or more output Transitions connected to it. Each Transition may or may not have an Event associated with it. If a Transition does not have an Event associated with it, then the Process fires the Transition immediately if it is enabled. If the Transition does have an Event associated with it, then the Process waits until the Action produces such an Event. When an Action-generated Event is received, the Process checks its eventcode attribute against the Events associated with every Transition. The Process will attempt to fire each Transition having the same eventcode associated with it, if that Transition is enabled. The Transition will be fired only for the color of the Action-generated Event.

When a Transition fires for a given color, it removes one Token of that color from each of its input Places, and it inserts a new Token with the same color into each of its output Places. When the Process is notified that a Token has been removed from a given Place, it automatically stops the Action (if any) associated with that Place. For this purpose, the Process invokes the Action's Stop() method passing as input parameter the color of the Token that has just been removed.

While a Transition fires, it creates a temporary Property container and a temporary Document container. In these temporary containers, the Transition combines all Properties and Documents from the Tokens it removes and from the Event (if any) that triggered the Transition's firing. The Transition combines these objects' Properties and Documents in the following way:

- The Transition picks one Property at a time from each Token, and adds it to the temporary Property container, discarding duplicate Properties (i.e., Properties with the same name). After all Tokens' Properties have been collected, the Transition will turn to the Event's Properties. For each Property in the Event, the Transition checks if there is any Property in the temporary container whose evaluate attribute contains the name of the Event's Property. If so, the Process assigns the value of the Event's Property to the Property in the temporary container. Furthermore, the Transition adds all Event's Properties to the temporary Property container, replacing any Property with the same name.

- The Transition picks one Document at a time from each Token, and adds it to the temporary Document container. Then, the Transition adds the Event's Documents to the temporary container. In both operations, documents with the same name are allowed and they are all added to the temporary Document container.

Only after the Transition has completed these operations will it remove the Tokens from its input Places, and insert new Tokens into its output Places. The new Tokens will be loaded with the Properties and Documents from the temporary containers. When these Tokens arrive at the output Places, the Process will transfer their Properties and Documents to the Places' associated Actions, and it will start those Actions. The whole behavior is then repeated.

7.1.5 Implementing the Workflow Kernel

The Workflow Kernel comprises a set of object classes that external application code can manipulate in order to define and run workflow processes. Besides a set of external interfaces, the Workflow Kernel specifies an event notification mechanism, which allows bidirectional interaction with external application code. In other words, external applications are able both to (1) invoke Workflow Kernel objects and to (2) be invoked by Workflow Kernel objects. Making use of these two possibilities, it should be possible to augment the Workflow Kernel with further functionality without having to modify it. The Workflow Kernel could then be provided as a pre-built software package.

7.1.5.1 Choosing the implementation technology

Given the purpose and approach of the Workflow Kernel, it becomes clear that it should be implemented with a distributed object middleware technology, namely COM, CORBA or Java RMI. There are some comparison studies for these technologies [Raj, 1998; Tallman and Kain, 1998; Chung et al., 1997], which highlight their differences and similarities. Regarding the implementation of the Workflow Kernel, COM seemed to be the most advantageous due to the following reasons:

- *application scenario* - The Workflow Kernel is a workflow enactment service, which is an inherently centralized component within a workflow management system. Therefore, the Workflow Kernel will be embedded either in a centralized application or in a single node of a distributed application. Furthermore, the Workflow Kernel will be have to be integrated with desktop applications such as modeling and monitoring tools. Therefore, the Workflow Kernel lends itself more to a single-system, desktop component architecture such as COM, rather than to a distributed component architecture such as CORBA, which excels in heterogeneous environments.

- *reference counting* - In COM, IUnknown is the base interface for every other COM interface. (In CORBA, there is similar base interface which is called CORBA::Object.) One of two fundamental features of IUnknown is that it allows a client object to query for a specific object interface from the set of all interfaces that a COM object implements. A client object can obtain a pointer to a specific interface by means of IUnknown's QueryInterface() method.

  The other fundamental feature of IUnknown is that it specifies a basic reference counting mechanism that all object interfaces must support. IUnknown's reference counting mechanism is based on two IUnknown methods: AddRef() and Release(). When an application obtains an interface pointer it must call AddRef() on that interface, and when the application is finished with using the pointer it must call Release(). This allows the underlying operating system to know when a COM object is being used and when it is no longer needed.

  The advantage of having a mandatory reference counting mechanism such as this one can hardly be overstated. In PRONEGI, for example, there is a FuncOpsManager application (section 6.1.2.3 on page 288) whose main purpose is to control the life-span of FuncOper

objects. The FuncOpsManager requires client applications to explicitly destroy FuncOper objects, since it never knows if a FuncOper object is being used or not. A reference counting mechanism would allow the FuncOpsManager to destroy FuncOper objects when they are no longer needed. In DAMASCOS, the WfFacility used to create an activity handler for each activity instance. This behavior has unveiled a major problem, as discussed in section 6.2.2.9 on page 328. With a reference counting mechanism, the WfBB would be able to destroy activity handlers when they are no longer needed, or create user pool without requiring the WfFacility to explicitly do so.

Besides reference counting, COM also employs another garbage collection mechanism. If after calling AddRef() a client fails and terminates before calling Release() then the reference count will never reach zero, so the object will not be destroyed. To correct these situation, COM employs a pinging protocol to detect if clients are still active. This protocol requires clients to periodically send a message to the server, so that if a client fails to send a message within a certain period of time, the server can assume that the client is lost, so the reference count is automatically decreased. (This pinging protocol is taken care by the underlying operating system.) The Java Virtual Machine (JVM) includes more elaborate ways to deal with garbage collection [Venners, 2000], but an interesting advantage of COM garbage collection mechanisms is that they are independent of the programming language.

- *connection points* - COM specifies a mechanism that allows objects to implement observer pattern behavior - this mechanism is known as *connectable objects* or *connection points* [Microsoft, 1995]. A connectable object is a COM object that defines one or more callback interfaces, in addition to its own external interfaces. In COM terminology, these callback interfaces are referred to as *outgoing interfaces*, and they must be implemented by clients of the connectable object. In this case, the connectable object is the event source, whereas the client object is the event sink. The connectable object is required to (1) create one connection point object for each outgoing interface, and to (2) implement the standard IConnectionPointContainer interface, as suggested in figure 7.25.

  The purpose of each connection point object is to implement the standard IConnectionPoint interface, which allows a client object to subscribe to that connection point. By subscribing to a connection point, a client object specifies that it is interested in receiving notification events via the outgoing interface that corresponds to that connection point object.

  Figure 7.25 illustrates the sequence of function calls that takes place between connectable objects and their clients. First, the client invokes the connectable object's IConnectionPointContainer interface (step 1) in order to retrieve a reference to the connection object that corresponds to the outgoing interface that the client implements. For example, if the client implements the outgoing Interface1, then it will use the connectable object's IConnectionPointContainer interface to find the connection point object for Interface1.

  After having retrieved an IConnectionPoint pointer, the client invokes the standard Advise() method, passing a reference to itself as input parameter (step 2). The connection point object will add this reference to its internal list of event sinks, so that the next time a notification event is generated the connection point object will notify this new client as well (step 3).

- *aggregation vs. inheritance* - Both CORBA and COM support interface inheritance, i.e., it is possible to define an interface having another interface as base interface. The design of the

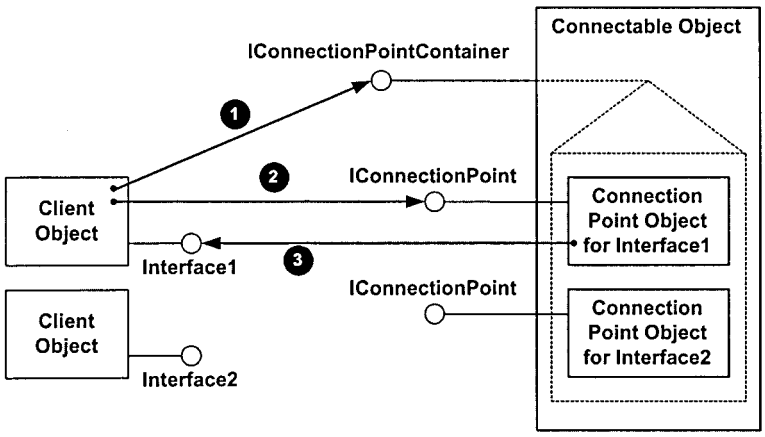


Figure 7.25: The COM connection points architecture

Workflow Kernel makes extensive use of interface inheritance, as illustrated in figure 7.18. As a consequence, all Workflow Kernel objects will have to implement their base interfaces' methods, and this means implementing the same behavior in several objects. In object-oriented programming languages such as C++, for example, this is usually not necessary since an object inherits not only the interface but also the implementation from its base class. However, neither CORBA nor COM address implementation inheritance.

Despite this fact, COM does provide an attractive alternative, which enables implementation reuse by means of *aggregation*. Aggregation allows an outer object to contain one or more inner objects, and to present the inner objects' interfaces as if they were implemented by the outer object, as suggested in figure 7.26. When a client asks the outer object for a pointer to an inner object's interface, the outer object returns an object reference to the inner, aggregated object. The client then invokes interface methods without being aware that they are actually implemented by the inner object, rather than by the outer object.

Aggregation faces two challenges. One is that a correct reference count must be maintained, so that both the inner and the outer objects are destroyed at the same time. The second challenge is that the inner object must be aware that it is being aggregated by an outer object, so that it can return a pointer to one of the outer object's interfaces, if requested to do so by a client. Both requirements can be met if the aggregated object delegates all calls made on its IUnknown methods to the IUnknown methods of the outer object, as suggested in figure 7.26. This assumes that a pointer to the outer object's IUnknown interface is supplied to the inner object when the inner object is being created.

Aggregation can effectively replace inheritance in the Workflow Kernel. For example, a Place object has several containers: an ITokenContainer, an IInputTransitionContainer, and an IOutputTransitionContainer. All of these containers implement the same basic Enumerator behavior. Therefore, instead of replicating this behavior, the Place object could just aggregate three Enumerator objects, one for each of its containers. Actually, it should be noted that if IPlace were to inherit the methods from ITokenContainer, IInputTransitionContainer and IOutputTransitionContainer this would lead to method ambiguities since all the three containers inherit their methods from the IEnumerator base interface. Aggregation can also replace the inheritance

relationship with IFeatures: all objects implementing this interface can simply aggregate a Features object that implements IFeatures instead of implementing those methods themselves.

- *implementation support* - In spite of the previous advantages, it is clear that the use of COM-specific features, such as reference counting and connection points, requires a significant amount of what is usually referred to as "plumbing code" or "boilerplate code". Reference counting requires all objects to implement IUnknown and to maintain an internal reference count; connection points require objects to implement some standard interfaces such as IConnectionPointContainer and IConnectionPoint; and aggregation requires IUnknown method calls to be delegated to the outer object's IUnknown interface. The COM Active Template Library (ATL) [Rector and Sells, 1999] relieves software developers from the burden of having to deal with these technology-specific details.

  ATL is a C++ template-based framework that contains a lot of pre-coded functionality to ease the development of COM objects. Basically, ATL includes (1) classes that wrap around interface pointers and basic data types, (2) classes that implement standard COM interfaces such as IUnknown, IConnectionPointContainer and IConnectionPoint, and (3) classes that implement COM functionality such as aggregation, object registration, and different threading models. In addition, ATL encourages the development of small-footprint, high-performance COM objects.

  ATL supports all of the previously discussed features: the ATL CComObjectRootEx class provides a thread-safe implementation of IUnknown; the CComObjectRootBase base class contains a pointer to an optional outer object, in case the object is being aggregated, and it implements IUnknown method delegation to that outer object; the IConnectionPointImpl and IConnectionPointContainerImpl classes implement the IConnectionPoint and IConnectionPointContainer interfaces, respectively. Typically, ATL promotes the reuse of this functionality via class inheritance: the C++ class that implements a COM object may have several ATL base classes.

- *security model* - COM makes use of existing security capabilities without requiring additional development efforts. Within a single host, COM relies on security capabilities provided by

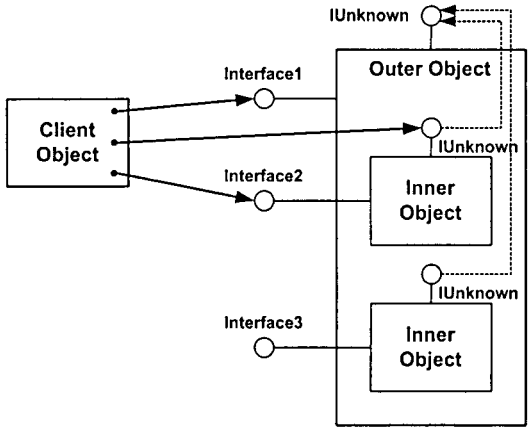


Figure 7.26: Reusing COM objects by aggregation

the operating system, which are common to all desktop applications. For Windows NT/2000, these capabilities are based on *access tokens* [Bates, 1999] (not to be confused with Petri net tokens): every system process (not to be confused with a workflow process) is given the access token created when the user logged on to the system, as shown in figure 7.27(a). The access token represents the rights the user has on that host, and it restricts the user's access to files and applications.

Distributed COM (DCOM) is the extension of COM technology to support distributed systems. In this case, the available security capabilities are based on Kerberos [Neuman and Ts'o, 1994], which is a state-of-the-art public key infrastructure (PKI). Kerberos allows a client and a server (also called the *verifier*) to authenticate each other and to communicate securely. Figure 7.27(b) illustrates how this happens: first, the client authenticates itself to an authentication server, which issues a certificate (called a *ticket*) for that client (steps 1 and 2). The authentication server also issues a *ticket granting ticket* (not shown in step 2): the client can use this ticket granting ticket to subsequently obtain new tickets (steps 3 and 4) without having to authenticate again with the authentication server. The client sends the ticket within the application request to the verifier (step 5). If mutual authentication is required, the verifier returns an authenticated application response.

7.1.5.2 The Workflow Kernel modules

The Workflow Kernel implementation has been divided into three modules: WfBase, WfKernel and WfAction. The WfBase module is a COM in-process server, i.e., a DLL intended to be attached to the WfKernel module. The WfBase module defines common Workflow Kernel interfaces: IFeatures, IEnumerator and all of the container interfaces shown in figure 7.17. It also defines INotifySink and the two enumeration types (obj_type and ntf_type) discussed in section 7.1.4.5. In addition,

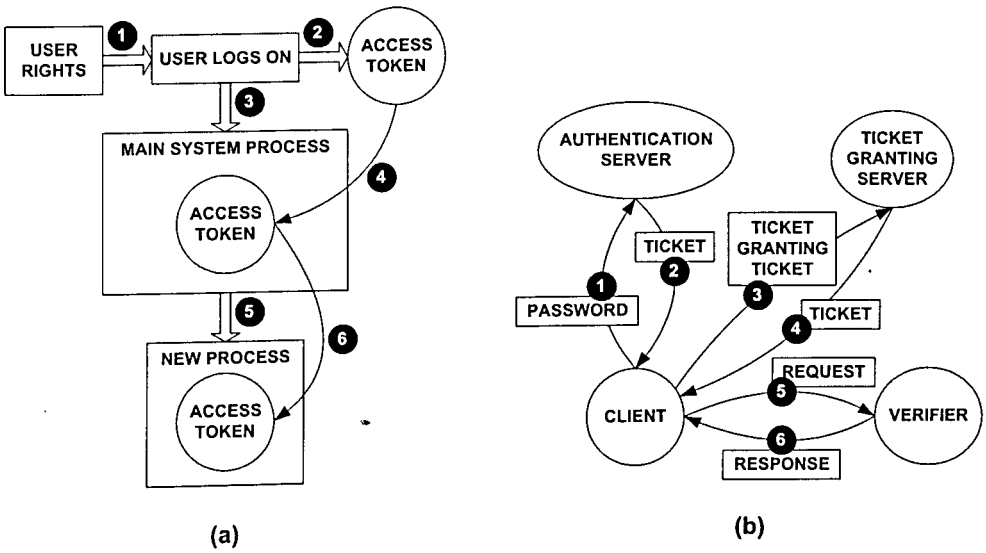


Figure 7.27: Security in Windows NT/2000 (a) and Kerberos (b)

the WfBase module implements two aggregatable COM objects - Features and Enumerator - which implement the IFeatures and IEnumerator interfaces, respectively.

The WfKernel module is the central module in the Workflow Kernel. It has been implemented as a Windows NT/2000 service, and it can run both as a local COM server and as a remote COM server via DCOM. The WfKernel module defines the object-specific interfaces presented in section 7.1.4.2, and it implements their corresponding objects, except for the Action object. The WfKernel module implements the following COM objects: Manager, Process, Place, Transition, Token, Event, Property and Document. The Manager object is implemented as a singleton COM object, so every client obtains the same object reference when it attempts to create a new Manager object. Implementing a singleton COM object with ATL is as simple as including the DECLARE_CLASSFACTORY_SINGLETON() macro in the C++ class definition.

The WfAction module is a placeholder for Action objects, which depend on the particular workflow management system the Workflow Kernel is built into. The WfAction module is not a mandatory part of the Workflow Kernel, since it is possible to build and execute processes without performing any concrete activity. However, it is certainly required for a particular workflow management system to implement its own Action objects, since these objects are a software layer between the workflow enactment service and the invoked resources. Action objects can be implemented outside the WfAction module, as long as they implement the IAction interface. Like the WfKernel object, so too the WfAction module has been implemented as a Windows NT/2000 service. It defines two enumeration types - one for IAction's type attribute and another for IEvent's eventcode attribute - and it includes some sample Action objects that have been used during the development and testing of the Workflow Kernel. These sample objects can be used as a starting point for the implementation of application-specific Actions.

The use of COM-specific features, especially aggregation and connection points, has had some impact on the original Workflow Kernel design. As suggested in the last section, all inheritance relationships - except the relationship between IEnumerator and the remaining container interfaces - have been replaced by aggregation. For this purpose, the WfBase module provides the two COM objects - Features and Enumerator - which the remaining Workflow Kernel objects will aggregate. According to the original inheritance-based design, all objects were required to implement the IFeatures interface; but now, according to the aggregation-based implementation, they are required to aggregate one Features object instead. Regarding container interfaces, these interfaces are all equivalent to the IEnumerator base interface, as shown in figure 7.17, so any of these container interfaces can effectively be implemented by an Enumerator object. Therefore, each container interface is implemented by aggregating an Enumerator object.

7.1.5.3 Aggregation of Enumerator objects

The aggregation of Enumerator objects has been implemented in the following way: (1) each Enumerator object has every container interface as an external interface, but (2) only one of these container interfaces is effectively exposed by aggregation. This is illustrated in figure 7.28: while each Enumerator object can be invoked through any container interface, only one of these interfaces is effectively available as an external interface for the outer object. In the particular example shown in figure 7.28, a client can obtain an ITokenContainer pointer, an IInputTransitionContainer pointer or an IOutputTransitionContainer pointer by querying the Place's IUnknown; but querying the Place's IUnknown for other container interfaces will fail. It should be noted that a client object cannot get a pointer

to other container interfaces even if it invokes the Enumerator's IUnknown interface because, as explained before, an aggregated object delegates all IUnknown calls to the outer object's IUnknown.

Internally, the C++ class that implements the Enumerator object contains a vector structure from the Standard Template Library (STL). This vector keeps the Enumerator's collection of object references. Inside the vector, the object references are stored as pointers to IUnknown interfaces. The main external interface of Enumerator is IEnumerator: this interface allows access to the object's internal vector. But, at the same time, the Enumerator must provide access to the same internal vector through any of the other specialized container interfaces. This is achieved with appropriate entries in the COM map, which ATL creates for each COM object. A COM map specifies all the interfaces exposed by an object and, typically, it contains one or more entries in the form:

```
COM_INTERFACE_ENTRY(x)
```

where *x* is the name of an interface that the COM object exposes. In the case of Enumerator, its COM map has only one of these entries and it looks like:

```
COM_INTERFACE_ENTRY(IEnumerator)
```

This effectively makes the Enumerator object expose the IEnumerator interface. In order to expose the specialized container interfaces, the Enumerator's COM map makes use of another kind of macro called COM_INTERFACE_ENTRY_IID(). This ATL macro takes two parameters:

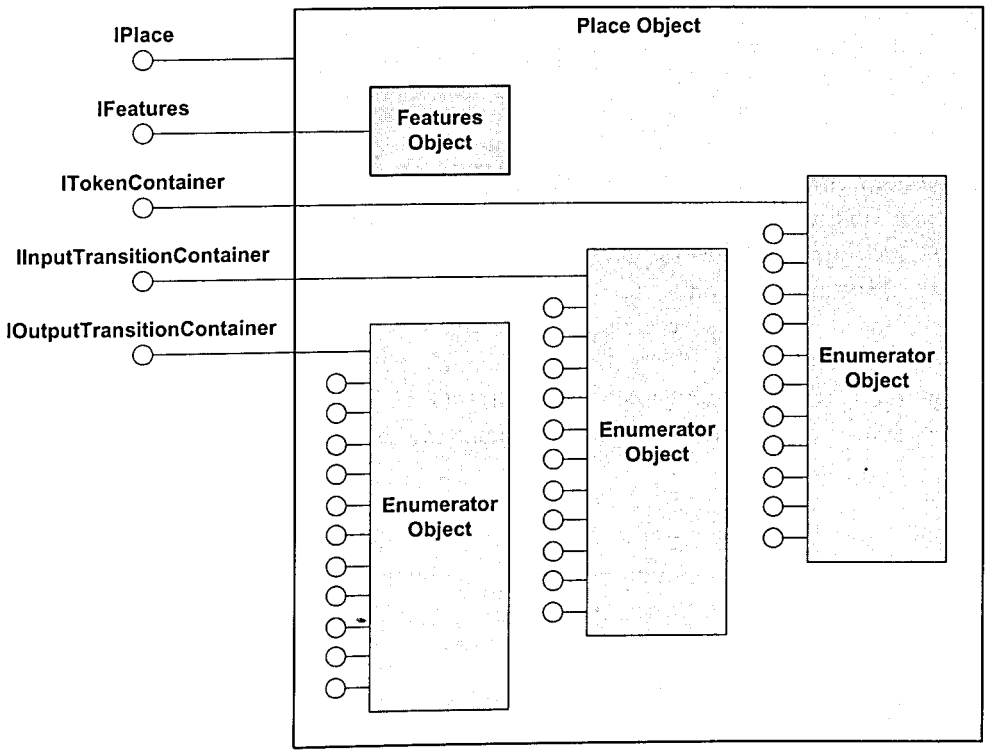


Figure 7.28: Aggregated objects within a Place object

COM_INTERFACE_ENTRY_IID(*iid, x*)

where *iid* is the globally unique identifier for the interface that is being exposed, and *x* is the name of the COM class which implements that interface. In the case of Enumerator, its COM map includes the following entries:

```
COM_INTERFACE_ENTRY_IID(IID_IPropertyContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IDocumentContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IActionContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_ITokenContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IPlaceContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IInputPlaceContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IOutputPlaceContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IEventContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_ITransitionContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IInputTransitionContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IOutputTransitionContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IProcessContainer, IEnumerator)
COM_INTERFACE_ENTRY_IID(IID_IColorEnumerator, IEnumerator)
```

Basically, what is being stated is that the Enumerator object exposes all of the specialized container interfaces, but if a client queries for any of these interfaces, it will get an IEnumerator interface pointer, i.e., an interface pointer to the same implementation as that of IEnumerator. Because the specialized container interfaces have exactly the same signature as IEnumerator, the client does not notice that it is actually being given an IEnumerator pointer rather than a pointer to the specialized container interface it queried for.

7.1.5.4 Internal notification of events

As suggested in figure 7.23, the following Workflow Kernel objects are event sources: Manager, Process, Place, Transition, Token, Action, Event, Property and Document. And as shown in figure 7.21(b), there are several notification events that these objects are able to produce. As a rule of thumb, every time the invocation of an interface method causes some change on the object that implements that interface, this object will generate a notification event. This event is notified to all objects which (1) implement the INotifySink interface, and which (2) have subscribed to the events generated by that event source. The event is also notified to the subscribers of other event sources which contain the first event source and which receive the notification event via the event flow mechanism depicted in figure 7.23.

However, each object aggregates one Features object and one or more Enumerator objects, which implement the IFeatures interface and the container interfaces, respectively. The outer object does not know how the aggregated objects implement their interfaces; what the outer object knows is that it exposes those interfaces as if they were some of its own interfaces. Therefore, although the outer object does not actually implement those interfaces, it is still responsible for notifying changes occurring within the aggregated objects. This problem can be illustrated by the example shown in figure 7.28: if a client object adds a new Token to a Place object (by means of ITokenContainer) then the Place object becomes responsible for notifying its subscribers that it was given a new Token, even though the ITokenContainer interface is actually implemented by a separate object (an aggregated Enumerator).

The solution to this challenge is to make aggregated objects notify their outer objects of the changes they produce. This is indeed straightforward to implement, since every aggregated object has a reference to its outer object: the pointer to its outer object's IUnknown interface. But how exactly should the aggregated object notify its outer object that a change has occurred? The answer to this question is simple, once one realizes that all outer objects (Manager, Process, Place, Transition, Token, Action, Event, Property and Document) implement the INotifySink interface. Therefore, an aggregated object can make use of its outer object's INotifySink interface in order to deliver events, as suggested in step 2 on figure 7.29.

7.1.5.5 The use of Connection Points

COM specifies an observer pattern mechanism called connection points, and ATL provides an implementation of this mechanism. This means that the observer pattern can be built into the Workflow Kernel with minimal effort. This mechanism is not required for the WfBase module, which implements Features and Enumerator, since these objects make use of a simple method call in order to notify their outer objects, as shown in step 2 on figure 7.29. But connection points will be required for the WfKernel and WfAction modules, which implement objects that must be able to notify each other as well as external subscribers, as shown in step 3 on figure 7.29. These objects are: Manager, Process, Place, Transition, Token, Action, Event, Property or Document.

Each of these objects plays the role of a connectable object, as shown in figure 7.25. Therefore, all of them must implement the standard IConnectionPointContainer interface. In practice, the C++ class that implements each of these objects just inherits from the IConnectionPointContainerImpl class, which is provided by ATL. The ATL library also implements the required connection points. The outgoing interface in this scenario is INotifySink, which client objects must implement to be able to receive events. To start receiving events, client objects can invoke the ATL function called

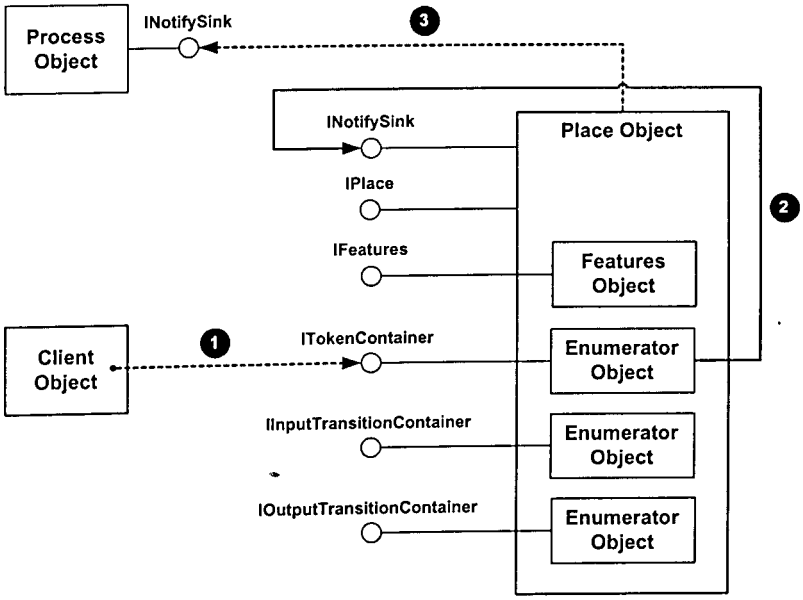


Figure 7.29: Internal event notification to the outer object

AtlAdvise(), which establishes a connection between a connection point and a client's event sink. When a client is no longer interested in receiving notification events it may invoke AtlUnadvise(), which terminates the connection.

To enable the event flow mechanism depicted in figure 7.23, each object must forward events coming from other objects that it contains, besides producing its own events. Thus, each object must be simultaneously an event source and an event sink, i.e., it must implement INotifySink too - which is precisely the interface it provides connection points for. This means that each Workflow Kernel object not only implements a connection point mechanism, but it must also behave as a client of that same mechanism. For example, both Processes and Places are event sources, so both of them implement connection points for INotifySink; but a Process is also an event sink for events generated by Places, so it implements INotifySink and receives events from the Places' connection points for INotifySink.

Figure 7.30 illustrates the use of connection points. In this example, it is assumed that there is initially just a single, empty Process object. When a new Place is added to the Process, the Enumerator object (which holds the collection of Places for that Process) automatically calls AtlAdvise() in order to establish a connection between the Place's connection point and the Process's event sink (step 1). When an Action is associated with that Place, the Place automatically calls AtlAdvise() in order to establish a connection between the Action's connection point and the Place's event sink (step 2). Actually, these two first steps cause two events to be generated, which are not shown in figure 7.30 for simplicity:

- The Process's Enumerator object produces a call of OnNotify(null, ENM_TOKEN, ITokenContainer, NTF_ADDELEMENT, IFeatures) on the Process's INotifySink interface.
- The Place object produces a call of OnNotify(null, TYP_PLACE, IPlace, NTF_ACTION, IAction) on the Process's INotifySink interface.

Both events are forwarded to other subscribers via the Process's connection point for INotifySink, after the Process object replaces the first parameter (null) with a valid pointer for its IProcess interface.

Now, if a client object adds a new Property to the Action, for example, then the Action's Property Enumerator generates a new event and notifies this event to its outer object with a simple method call (step 3). But all Workflow Kernel objects must forward events, so the Action object delegates the call to its connection point (step 5). In this example there is only one event sink, which is the Place object, so this object is notified of the event (step 6). Again, the Place object delegates the event to its own connection point (step 7), which will forward it to the corresponding event sinks (step 8). And once the event reaches the Process object, it is again passed on to the Process's connection point (step 9), and from this connection point on to other subscribers.

7.1.6 Reusing the Workflow Kernel

To demonstrate how the Workflow Kernel can be useful and to illustrate the required effort to embed such a reusable workflow enactment service in a particular workflow management system, one can turn to the PRONEGI and DAMASCOS projects, which have been presented in the previous chapter. On one hand, these are not the ideal testing examples because some of the requirements for the Workflow Kernel have been derived precisely from those projects, so the Workflow Kernel may fit better to these workflow management systems than to others. On the other hand, the fact that PRONEGI and DAMASCOS have been thoroughly described makes them the best candidates to illustrate the use of

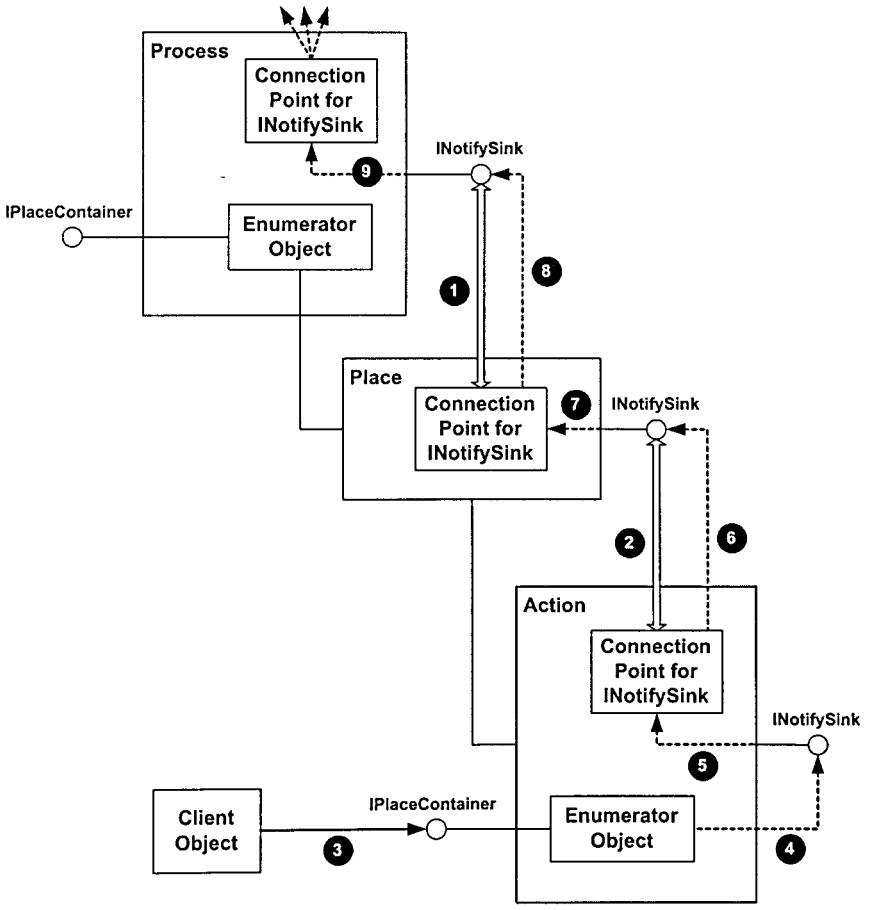


Figure 7.30: The event flow mechanism with COM connection points

the Workflow Kernel, and one can expect that the developers of other workflow management systems will be able to figure out similar or additional ways in which the Workflow Kernel can be reused in their systems.

7.1.6.1 Potential use in PRONEGI

Figure 7.31(a) illustrates how PRONEGI processes can be modeled with the Workflow Kernel. In this case, each pair of Place and Transition represents a single activity, which is to be performed under a certain Context. The XFlow Process Editor may still represent activities by means of blocks, as in figure 6.15 on page 298, although internally each of these activities now corresponds to a Place associated with an Action and connected to a Transition associated with an Event. Some auxiliary Places may be required in order to come up with a sound WF-net. When the user drags a functional operation and drops it onto an activity, as in figure 6.15, it is actually doing three things: (1) it is creating a new Place and a new Transition, (2) it is associating an Action with the new Place, and (3) it is setting the value of an Action's Property which contains the name of a FuncOper object. When

the user drags a Context and drops it onto the activity, it is setting the value for another Property which contains the name of the Context under which the activity should be dispatched.

The input data parameters shown in figure 6.16 on page 299 are also analogous to a set of Action's Properties, whereas the output parameters represent the Properties that will be brought back to XFlow by the Action-generated output Event. According to the process model shown in figure 7.31, an Action must be a software component that communicates over the WfBB in order to implement the activity assignment protocol shown in figure 6.20 on page 303. The Action publishes an activity request under the specified Context, it waits for the first workflow client to reply and then it assigns the activity to that client. After this assignment phase, the same Action waits for some kind of response from the functional operation, and as soon as the response is received, it generates an Event object and communicates this Event back to its Place, which will forward it to the Process object and so on, according to the event sequence depicted in figure 7.24.

Every Process has a special input Place, as shown in figure 7.31, but which may not be explicitly represented in the process model. Anyway, when the user sets the process triggering condition, as shown in figure 6.17(b) on page 300, it is effectively setting the Event that is associated with the first Transition in the Process. When this Event occurs, the Transition fires and the Process runs. This assumes that a second kind of Action is associated with the Process's input Place. This Action does not dispatch any activity request, it just listens for triggering messages coming on a special-purpose Context called "XFlowTrigger", as explained in section 6.1.3.3 on page 299. Since every Process has an input Place and a triggering Transition, the XFlow application should automatically include these elements in every process definition, without requiring the user to do so explicitly.

Most of the existing graphical user interfaces could still be used as the front-end for a new XFlow application based on the Workflow Kernel. In fact, this new XFlow application would have to: (1) implement the PRONEGI-specific Actions just described, and (2) provide the graphical user interfaces. The Workflow Kernel would store process definitions and would provide process enactment capabilities. To the user, the new XFlow application could look just like the previous one. However, the way of creating process instances is now totally different: instead of duplicating the process definition, the Workflow Kernel uses the same Process object with distinct Token colors. As explained in section 7.1.3.3, the Token color must go all the way to the functional operation, and then backwards from the functional operation to XFlow. Therefore, a new parameter - the Token color - would have to be

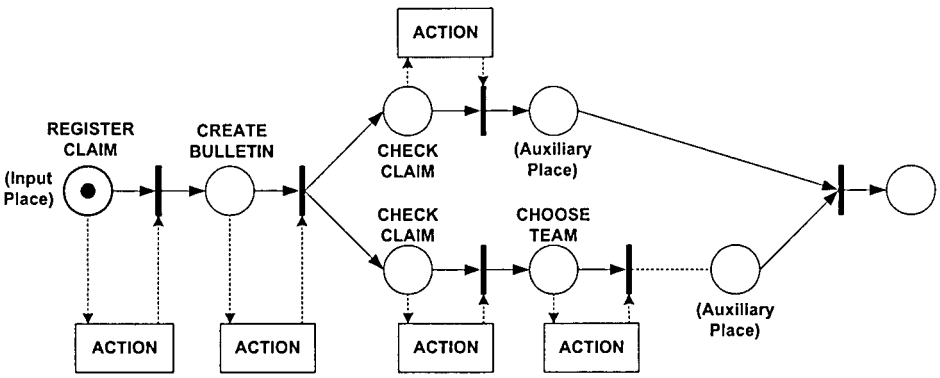


Figure 7.31: Modeling the TR/D process with the Workflow Kernel

added to the invoking URL shown on page 302, and the functional operations should be prepared to return this value as an output parameter when their operation completes.

7.1.6.2 Potential use in DAMASCOS

In DAMASCOS, the Workflow Kernel could provide the required process enactment capabilities, and it could completely replace the OMG Workflow Management Facility implementation. In fact, the Workflow Kernel's modeling approach - especially the use of Actions and Events - would be a perfect fit for the document exchange requirements between business functions. Figure 7.32 illustrates how the Workflow Kernel could be used in the DAMASCOS WfFacility. Basically, each Action is implemented by an activity handler that sends an XML document via the WfBB to a given business function. This business function is specified as an Action's Property that contains the name of a WfBB user. If a reply should be expected from the business function, then an Event must be associated with the following Transition; otherwise, if no Event is associated, the Process proceeds immediately without waiting for a reply. This effectively supports activities that produce an output document as well as those that do not produce any output.

As explained in sections 6.2.2.6 and 6.2.2.7 on page 324, the default push user reads a configuration file that contains a set of process triggering conditions. In that case, one of the elements for each triggering condition was the name of a WPDL file which contains a description of the process to be triggered. Now, with the Workflow Kernel, this element would have to be replaced by the name of the Process object to be started. When the default push user receives a certain document type from a

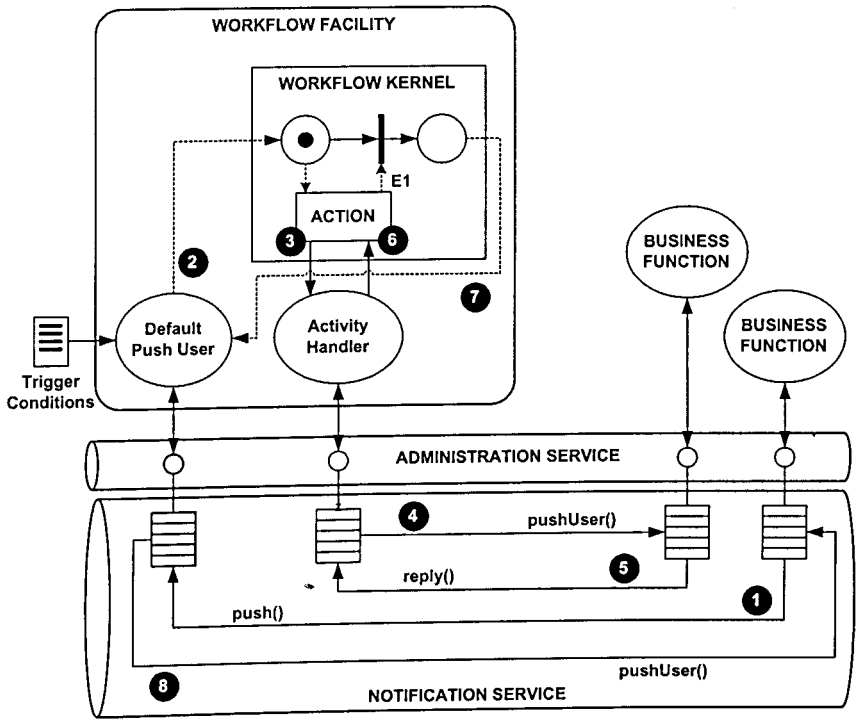


Figure 7.32: A new WfFacility based on the Workflow Kernel

certain WfBB user (step 1 in figure 7.32), it reads its configuration file and determines which Process should be started. Then it obtains, from the Manager object, a reference to the Process object.

With this Process reference, the default push user sets the received document as a Process's Document. This is similar to what the WfFacility already did in the former case, by setting the received document as the input data for the process instance. The default push user then invokes the IProcess's Start() method: as a result, a new Token will be inserted into each Process's input Place (step 2), and this Token will be loaded with the Process's Document. When the Token arrives at a Place with an associated Action, it transfers the Document to the Action, so the Action now has one Document (the one just transferred from the Token) and one Property that identifies the intended recipient for the Document. The Action's Start() method is then invoked.

The Action then creates an activity handler that will send the Document to the assigned WfBB user (steps 3 and 4), and it will wait for a reply from that WfBB user. However, if the following Transition has no associated Event, it will fire immediately and the Action will be stopped. If the Transition does have an associated Event, then this means that the Process will wait until the Action generates such Event. This will happen when the WfBB user replies with an output XML document (steps 5 and 6). After that, Process execution proceeds with the following Actions until the Process's output Place is reached. At this point, the Process generates a special Event (NTF_PROCESSCOMPLETE), which the default push user will react to by returning the last output Document back to the WfBB user which triggered the Process (steps 7 and 8).

An important issue regarding the use of the Workflow Kernel in DAMASCOS is that the Token color cannot actually travel as a parameter to the business function. This is due to the fact that DAMASCOS business functions are only required to produce and consume application-specific XML documents, and that the WfBB and the WfFacility have no influence in the definition of these document formats. Therefore, the WfFacility cannot send a color parameter and expect the business function to return the same parameter; it must rely on another mechanism to correctly identify which activity an incoming message is related to. This is possible if there is a unique activity handler for each activity instance, as it happened before. In this case, each activity handler only receives messages concerning a single Token color.

7.1.6.3 Modeling and monitoring tools

Both PRONEGI and DAMASCOS have devised tools to model and monitor workflow processes. Clearly, the implementation of such tools depends on the data structures that each workflow management system uses to represent processes internally. In PRONEGI these data structures are completely hidden inside the XFlow application, whereas in DAMASCOS they can be accessed via the CORBA interfaces defined by the OMG Workflow Management Facility specification. In either case, if the workflow enactment service is replaced by the Workflow Kernel, these data structures will change completely because the Workflow Kernel introduces its own set of objects and interfaces. Therefore, the modeling and monitoring tools have to be rewritten.

Despite this fact, the Workflow Kernel allows modeling and monitoring tools with varying degrees of sophistication to be developed. Modeling tools will invoke Workflow Kernel objects via their interfaces in order to define Processes and the Actions and Events associated with Places and Transitions, respectively. Monitoring tools will make use of the Workflow Kernel's event notification mechanism, i.e., they will implement the INotifySink interface in order to be notified of changes in Workflow Kernel objects. In the Workflow Kernel, process modeling and process monitoring are almost analogous counterparts: just like an modeling application may invoke a Process's interface or

get into more detail by invoking a Place's interface, so too can a monitoring application subscribe to events from a Process or only from a Place within that Process.

An interesting possibility is that the Workflow Kernel allows an external application to do both things at the same time, i.e., to manipulate objects and to receive events from those objects simultaneously. Thus, it is possible to have a modeling tool that is also a monitoring tool, or the other way around. On one hand, this means that it is possible to have two modeling tools displaying the same Process, and the changes inserted via one modeling tool are immediately reflected onto the other, which is an interesting feature for a workflow management systems. On the other hand, it also means that process modeling and process monitoring no longer have a clear-cut gap; in fact, they may even overlap. For example, while a WF-net is being defined or refined, some of its process instances may be already running and they could be monitored within the modeling tool, which is another interesting feature for a workflow management system. This is possible because the process definition (the WF-net) is exactly the same data structure in which process execution is recorded, as discussed in section 7.1.2.3.

Figure 7.33 shows the visual appearance of two graphical components developed specifically for the Workflow Kernel. Since the Workflow Kernel is based on COM, these two components have been implemented as ActiveX controls, so they can be integrated with other applications that use this technology. In figure 7.33, these components are embedded in the Web browser by means of an HTML page. On the left-hand side there is the WfKMonitor component, which basically presents a tree with all Workflow Kernel objects and their contained objects. On the right-hand side there is the WfKEditor component, which allows Processes to be defined and edited, namely by adding or removing Places and Transitions, by connecting Places to Transitions, by associating Actions to Places and Events to Transitions, by configuring the Properties and Documents for Actions and Events, and by adding or removing Tokens to the Process.

Both of these components subscribe to events from the Workflow Kernel: the WfKEditor subscribes only to the events produced within the Process being edited, whereas the WfKMonitor subscribes to events from the Manager object, so the WfKMonitor effectively receives all events produced within the Workflow Kernel. As a result, any change introduced via the WfKEditor - or, as a matter of fact, via any other local or remote application connected to the Workflow Kernel - is immediately reflected onto the WfKMonitor. Additionally, if another application introduces a change to the Process being edited, this change is also immediately reflected on the WfKEditor.

7.1.6.4 Building workflow management systems

The Workflow Kernel provides the basic functionality of a workflow enactment service, as well as two kinds of interfaces that allow external applications to be integrated with it. The two kinds of interfaces are the object-specific interfaces described in section 7.1.4.2 and the callback interface described in section 7.1.4.4. Having these two kinds of interfaces, it is possible to build application code that either invokes Workflow Kernel objects or reacts to changes in these objects. Some components even combine the use of both kinds of interfaces.

The WfKEditor tool is an external component whose purpose is allow processes to be defined graphically. This component invokes object-specific interfaces in order to create and establish associations between Workflow Kernel objects; on the other hand, the WfKEditor component implements the INotifySink callback interface in order to be notified of any changes in the objects it has created. The Action objects illustrate another kind of integration: these objects are invoked by the Workflow Kernel whenever a new Token is inserted into a Place, and they notify the Workflow Kernel of any

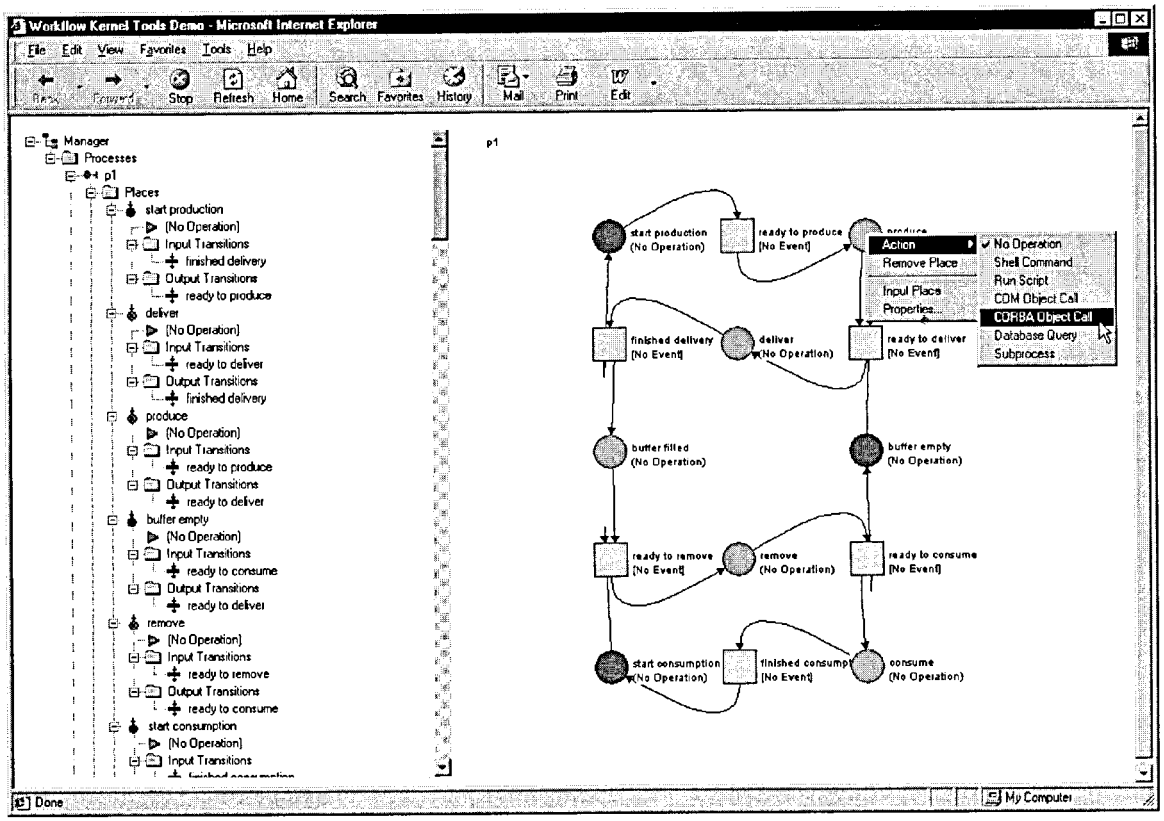


Figure 7.33: The WfKMonitor and the WfKEditor tool components

Action-generated Event. Therefore, each Action object implements an object-specific interface (IAction) and is able to invoke the INotifySink callback interface, now implemented by the Workflow Kernel itself.

The WfKEditor tool and the Action objects illustrate the fundamental difference between components that can be regarded as belonging to the Workflow Kernel (e.g. Processes, Places, Actions) and components that are external to the Workflow Kernel (e.g. WfKEditor, WfKMonitor): *internal components* implement an object-specific interface and invoke the callback interface, whereas *external components* invoke object-specific interfaces and implement the callback interface. In both cases, integration is bidirectional: there is always one interface that allows a component to invoke the Workflow Kernel, and another interface that allows the component to be invoked by the Workflow Kernel.

With these interfaces, it is possible to develop components that augment the functionality of the Workflow Kernel. Sections 7.1.6.1 and 7.1.6.2 discussed how Actions objects should be implemented for two particular workflow management systems; section 7.1.6.3 discussed how modeling and monitoring tools could be implemented. Ultimately, the functionality provided by the Workflow Kernel can be augmented until the result is a system that covers the entire functionality of a workflow management system. Figure 7.34 illustrates the resulting workflow management system architecture. Modeling and monitoring tools are external components which are integrated directly with the

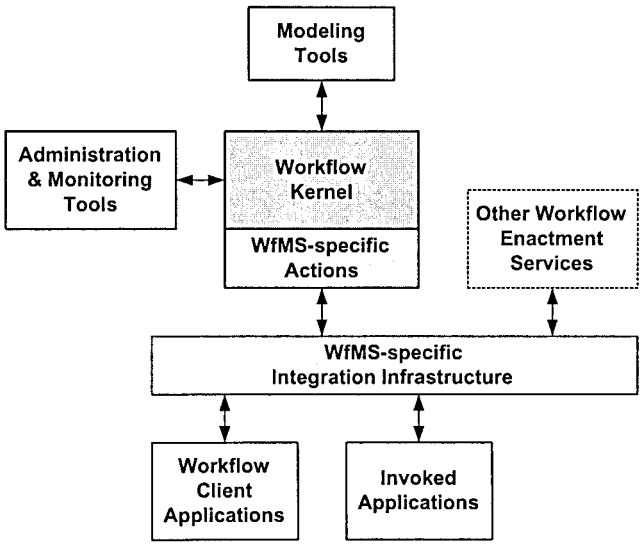


Figure 7.34: Proposed architecture for workflow management systems

Workflow Kernel. Action objects are internal components that implement the behavior of workflow activities over an integration infrastructure.

7.2 Integration infrastructure and interaction services

Now that the workflow enactment service has been described it is time to address the integration infrastructure, so that the combination of these two components results in a workflow management system that can support the engineering of business networks. In chapter 6 it was found that such a workflow management system requires a decentralized messaging infrastructure, i.e., something with similar capabilities to those provided by a MOM system, but in a completely decentralized fashion. Within an enterprise, a centralized infrastructure such as a MOM system is a suitable solution since all resources perform their activities under hierarchical control. However, an inter-enterprise environment calls for a radically different kind of solution since the resources to be integrated should be regarded as being autonomous or as being under the command of autonomous entities.

The need for a decentralized infrastructure should not be underestimated. In DAMASCOS, the development of a workflow-enabled messaging system - the WfBB - has resulted in a centralized infrastructure. This infrastructure was found to be a suitable solution since DAMASCOS assumed, from the beginning, that supply chains made up of SMEs require the coordination of actions which are initiated and controlled by the principal producer. In other words, the scenarios that DAMASCOS was built to address still fall under the scope of hierarchical control. But soon it was realized that such a centralized infrastructure had some limitations: the product manufacturer might need to integrate external, autonomous SMEs into its own supply chain, or it might want to participate in additional supply chains where it does not play the role of principal producer. In these cases, it is not appropriate to have the infrastructure centralized at a single node, so DAMASCOS has tried to come up with a more decentralized infrastructure by federating several WfBB instances. However, it was quickly

found that there was no easy way to do this without reinventing the functionality that the WfBB already provides.

This experience from the DAMASCOS project illustrates a fundamental point: the fact that a decentralized infrastructure cannot be built by extending a centralized one. Therefore, it is imprudent to think of developing a decentralized integration infrastructure with technologies that rely on centralized services. To support the engineering of business networks, the desired integration infrastructure should provide message exchange services within a network of autonomous entities, where no single entity can own or control these services. Each enterprise brings into the network its own services and its willingness to interact with services provided by other enterprises. Enterprises should then be able to interact with the services provided by one another without any kind of centralized control, and without having to rely on - or being restricted by - third parties. Such an integration infrastructure must not based on the functionality of a single system, but on the compound of services that is achieved by connecting enterprises together. Only peer-to-peer networking can achieve such a feat.

7.2.1 Building a P2P integration infrastructure

Project JXTA, presented in section 5.5.5, provides an interesting platform to implement a decentralized integration infrastructure for a workflow management system supporting the engineering of business networks. An integration infrastructure built on top of JXTA allows enterprises not only to interact with each other but also to discover each other and to provide specialized services to each other. These possibilities are not fundamentally new, since they can be accomplished using existing approaches such as, for example, e-marketplaces. What is fundamentally new about using a P2P integration infrastructure is the fact that enterprises are able to discover and interact with each other without having to rely on centralized services provided by third parties. Thus, enterprises are able to integrate and synchronize their services directly with each other without any intermediaries. Furthermore, no single enterprise can have full control over the integration infrastructure: each enterprise comes with its own set of services, which interact with the services provided by other enterprises. In this context, "service" refers to any kind of software functionality that allows peers to interact with each other across the P2P network.

7.2.1.1 Multicast and unicast services

In a P2P infrastructure there are two kinds of services: multicast services and unicast services. Gnutella, for example, specifies a distributed search protocol that broadcasts queries across a P2P network, and it specifies a file download protocol that allows a peer to transfer files from another peer; the distributed search protocol plays the role of a multicast service, whereas the file download protocol plays the role of a unicast service. In JXTA there are also multicast and unicast services: the DiscoveryService, for example, is a multicast service because it allows a peer to propagate queries to other peers, whereas the EndpointService is a unicast service because it allows a peer to route messages to a specific remote endpoint.

With JXTA, there is more than one way to implement multicast and unicast services. At a lower level, it is possible to use EndpointService to implement unicast services, and use the ResolverService to implement multicast services. At a higher level, it is possible to use pipes to implement both kinds of services: unicast pipes support unicast services, and propagate pipes support multicast services. In a P2P integration infrastructure, it makes sense to a high-level abstraction such as propagate pipes to implement multicast services. Regarding unicast services, however, it is simpler to use endpoint

addresses rather than unicast pipes. The reason for this is that establishing a pipe require peers to exchange the corresponding Pipe Advertisement and to bind an input pipe and an output pipe, whereas endpoints are immediately available as soon as a peer connects to the P2P network.

The only issue about using endpoints is the question of how a peer can obtain an endpoint address of another peer. In general, the use of a unicast service is always preceded by the use of a multicast service. Therefore, in most cases the endpoint address can be supplied in a response to a previous multicast query. It is interesting to note that the use of pipes follows the same principle: before a peer can create an output pipe, it must obtain the Pipe Advertisement by invoking the DiscoveryService, which is a multicast service; and when the peer obtains the Pipe Advertisement it must invoke the PipeService, which is another multicast service, in order to find the peer that has the corresponding input pipe. Thus, either using endpoint addresses or pipes, the use of a unicast service always follows the use of a multicast service which allows a peer to determine the endpoint address of a remote peer it wants to communicate with.

Figure 7.35 illustrates the use of multicast and unicast services within a peer group. In figure 7.35(a) it is assumed that peers 2, 3 and 4 have created an input pipe based on a Pipe Advertisement that describes a propagate pipe. Peer 1 creates the output pipe based on the Pipe Advertisement and sends a message through that pipe; the message reaches all peers that have a corresponding input pipe. Having received the request message, the peers may or may not reply to the request, depending on

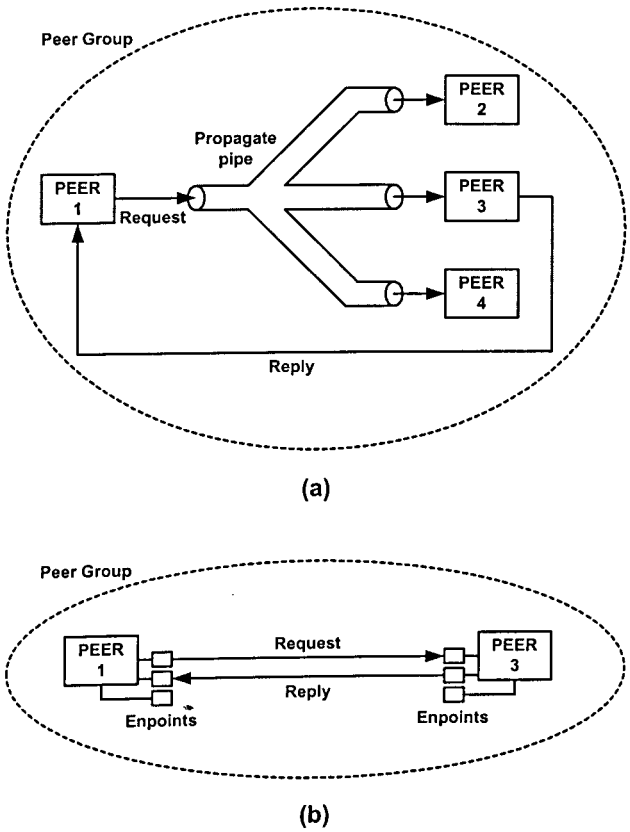


Figure 7.35: Multicast (a) and unicast (b) services within a peer group

the particular service being invoked and depending on whether each destination peer has some data that matches the request. In the example shown in figure 7.35(a) only one peer replies to the original message. This reply is a unicast message that can be addressed to an endpoint or sent through a unicast pipe to peer 1.

If endpoint addresses are used, peer 3 can obtain the endpoint address of the sender from the original request message. Alternatively, if a unicast pipe would be preferred, peer 1 would be required to publish an appropriate Pipe Advertisement or to include that Pipe Advertisement in the request message. Having received a reply from peer 3, peer 1 may want to communicate further with peer 3. For this purpose, it is clear that they must use a unicast service. If this unicast service would be based on pipes, then peer 3 is now assumed to have published an appropriate Pipe Advertisement or to have included that Pipe Advertisement in the reply sent to peer 1. Once again it is simpler to use endpoint addresses, since peer 1 could extract the endpoint address of peer 3 from its reply message, and immediately initiate a unicast conversation as shown in figure 7.35(b). The P2P integration infrastructure makes use of propagate pipes to implement multicast services, and it makes use of endpoint addresses to implement unicast services.

7.2.1.2 Main objects and interfaces

The P2P integration infrastructure has been developed as a software layer on top of JXTA; referring to figure 5.52, this software layer is positioned between the JXTA standard services layer and the JXTA community services layer. Its purpose is to allow peers (i.e., enterprises) to deploy their own services and to interact with services provided by other peers, in such a way that they make use of JXTA capabilities without having to be aware of JXTA-specific concepts such as advertisements and pipes. Basically, this software layer encapsulates the JXTA protocols and standard services and simplifies the access to a P2P, service-based integration infrastructure. Since the JXTA reference implementation has been developed in Java, the software layer has been implemented as a set of Java classes, which are depicted in figure 7.36. In order to facilitate the integration with external applications, these classes have been exposed as a set of CORBA interfaces, using the JDK ORB.

The Service class The most important class in the integration infrastructure is the Service class, which defines the basic features of a P2P service: a service has a name, it must be able to send and receive messages, and it must be able to notify other modules whenever a new message is received. Each single P2P service can be regarded from two complementary perspectives: from the service client side the service must be able to send requests and wait for replies, and from the service provider side the service must be able to receive requests and send replies. The Service class can represent both of these perspectives:

- A service client invokes the createRequest() method, which creates a ServiceMessage object representing a request, followed by the sendMessage() method in order to send that ServiceMessage. After the request is sent, the service client listens for incoming replies.

- A service provider invokes the startService() in order to start listening for requests. To produce a reply, the service provider invokes the createReply() method, which takes the original request message as input parameter and creates a new ServiceMessage object representing the reply. Then it invokes the sendMessage() method in order to send that ServiceMessage.

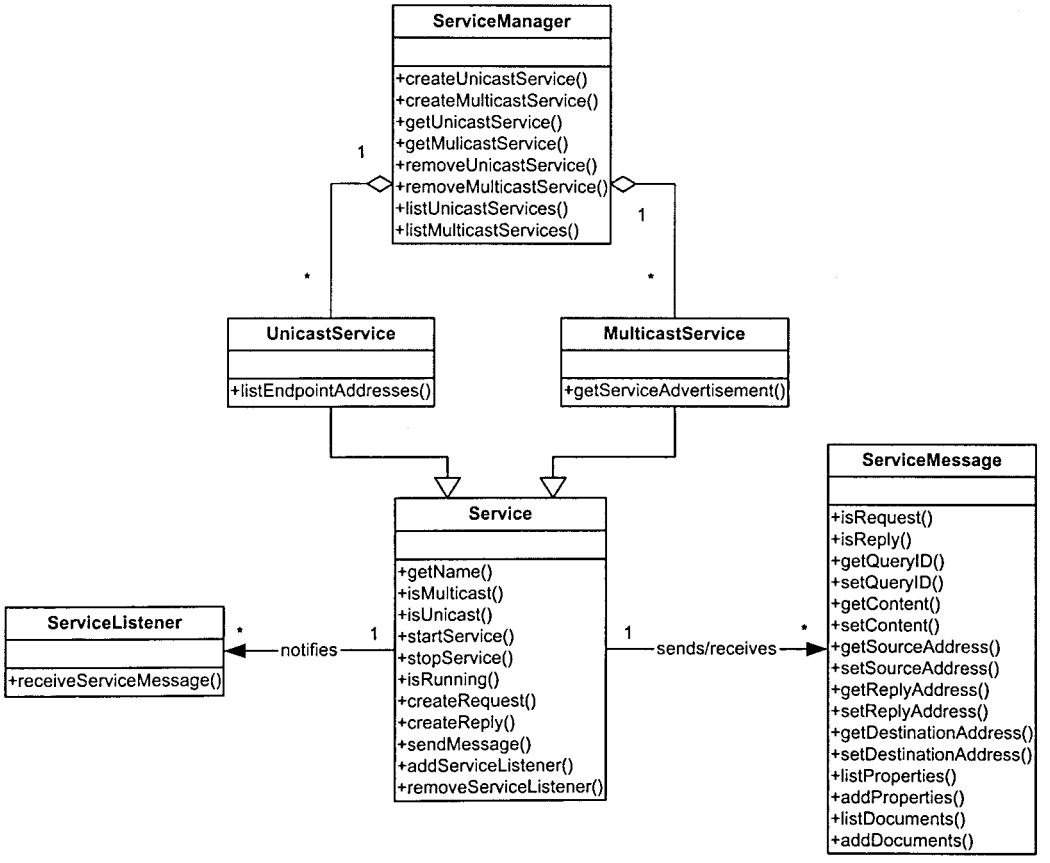


Figure 7.36: Main classes in the integration infrastructure

The UnicastService class The UnicastService class represents any service whose implementation is based on endpoint addresses. The startService() method is invoked by a service provider in order to start listening for incoming requests on one of its endpoints. It should be noted that a peer is able to receive endpoint messages even before the startService() method is invoked but, in that case, the incoming messages do not reach this software layer. It is only when startService() is invoked that the UnicastService object registers itself as an EndpointListener, so the EndpointService will notify the UnicastService of incoming messages. Actually, the startService() method registers the UnicastService object as an EndpointListener for all messages addressed to its own service name, and having the string "Request" as service parameter.

The stopService() method unregisters the UnicastService as an EndpointListener, so that it stops receiving requests. The isRunning() method returns true only if the startService() method has been invoked but the stopService() method has not.

From the service client perspective, when the sendMessage() method is used to send a request, this method automatically registers the UnicastService object as an EndpointListener for all messages addressed to its own service name, and having the string "Reply" as service parameter. Therefore, a service client is an EndpointListener too, but it listens for replies rather than requests.

For the UnicastService class, the isUnicast() method always returns true and the isMulticast() method always returns false.

The MulticastService class The MulticastService class represents any service whose implementation is based on propagate pipes. The startService() method is invoked by a service provider in order to start listening for incoming requests on an input pipe. First, startService() tries to find, using the DiscoveryService, the Module Specification Advertisement for a service with the same name as the MulticastService object. If startService() cannot find the Module Specification Advertisement, it creates and publishes the Module Specification Advertisement itself. Either way, the startService() method gets a Pipe Advertisement, which is used to create an input pipe. Then, startService() launches a new thread that will listen for incoming requests on that pipe. The stopService() method terminates this thread.

For each request message that the MulticastService receives, it may generate zero or more replies. To do this, it invokes the createReply() method, which extracts the endpoint address of the sender of the original request and sets it as the destination endpoint for the reply. Then, it invokes the sendMessage() method, which sends the reply to the destination endpoint via the EndpointService. Therefore, although the request is transmitted via the PipeService, the reply is sent back via the EndpointService.

From the service client perspective, when sendMessage() is used to send a request, the first thing this method does is to find the Module Specification Advertisement. Then, with the Pipe Advertisement that comes with the Module Specification Advertisement, sendMessage() creates an output pipe using the PipeService, and sends the request through this output pipe. At the same time, it registers the MulticastService object as an EndpointListener for all messages addressed to its own service name, and having the string "Reply" as service parameter.

For the MulticastService class, the isMulticast() method always returns true and the isUnicast() method always returns false.

The ServiceListener interface The ServiceListener is just a CORBA interface without a corresponding implementation class. It is callback interface to be implemented by external applications or modules, which may need to react to received messages or may simply have to be notified whenever a new message is received. For example, if a given service implementation relies on the functionality provided by an external module, this module has to be notified whenever a new request is received; on the other hand, if an external module is expecting a response to a previously sent request, this module should be notified as soon as the reply is received. Both of these scenarios require the P2P integration infrastructure to be able to notify external applications of incoming messages from remote peers.

In order to subscribe to messages received by a Service object, an application must implement the ServiceListener interface and it must invoke the Service's addServiceListener() method, passing a reference to a ServiceListener object as input parameter. The Service object maintains a list of all ServiceListener objects that it has to notify. When a Service object receives a message, either through a pipe or by means of endpoint addressing, the Service object creates a new ServiceMessage object and loads this ServiceMessage with all the information it can extract from the received message. Then, the Service object notifies each one of the ServiceListener objects by invoking their receiveServiceMessage() method, passing two input parameters: one is a reference to the Service object, and the other is a reference to the ServiceMessage object just created.

The ServiceManager class The ServiceManager class allows the peer to create, destroy or retrieve references to existing Service objects. Whenever a peer wants to become a service provider or play the role of service client, it must invoke the ServiceManager interface in order to create a UnicastService or a MulticastService object with a given name. If a Service object with the same name already exists, ServiceManager returns a reference to the existing object. UnicastService and MulticastService objects are managed separately, so it is possible to have a UnicastService and a MulticastService with the same name, but actually representing different P2P services. However, the ServiceManager does not allow two UnicastServices nor two MulticastServices to have the same name.

For each instance of the integration infrastructure running at a given peer there is only one ServiceManager object. This ServiceManager object is created by a ServiceManagerServer object, which is not shown in figure 7.36 because it does not have an external CORBA interface. Running a ServiceManagerServer object actually corresponds to launching an instance of the integration infrastructure. The ServiceManagerServer object accepts a command-line parameter that specifies the peer group to join when the peer enters the P2P network; all service interaction will take place within this group. The ServiceManagerServer also registers a reference to the ServiceManager object in the CORBA Naming Service, so that external applications may retrieve the object reference and use it to access the integration infrastructure. The name used to register the ServiceManager object in the Naming Service is the name of the peer group.

7.2.1.3 Service message structure

Regardless of their particular communication mechanisms, both UnicastService and MulticastService objects make use of ServiceMessage objects to exchange requests and replies. According to figure 7.36, the ServiceMessage class has several attributes: the QueryID attribute is a parameter that allows peers to relate replies with the original requests, the Content attribute is a general-purpose text field, the SourceAddress attribute contains an endpoint address of the peer that created the message, the ReplyAddress attribute contains an endpoint address to which replies should be sent, and the DestinationAddress attribute specifies the endpoint address that the message should be sent to. Usually, the ReplyAddress is the same as the SourceAddress and, in multicast services, the DestinationAddress is unused.

Besides these attributes, a ServiceMessage may contain an arbitrary number of properties and documents. The properties are expressed as strings having the form property_name = property_value, and the documents are expressed as strings having the form document_name = file_name, where file_name specifies the location of the document file (e.g., a local file path or a URL). When the ServiceMessage is passed as input parameter to sendMessage(), this method creates a suitable JXTA message and appends the ServiceMessage attributes to the JXTA message. The result is shown in figure 7.37. For a multicast message, to be sent through a pipe, a Rendezvous Propagate Message is created, and for an endpoint message an Endpoint Router Message is created. In both cases, the ServiceMessage attributes are appended as XML elements to the message.

Each ServiceMessage attribute has a corresponding element in the JXTA message. For some attributes the correspondence is quite straightforward: the MSG_FROM element contains the SourceAddress attribute, the MSG_REPLY_TO contains the ReplyAddress attribute, the MSG_QUERY_ID element contains the QueryID attribute, and the MSG_CONTENT contains the Content attribute. Only the DestinationAddress is missing, which is used to send the message via the EndpointService. The MSG_PARAM element specifies whether the message is a "Request" or a "Reply". The MSG_PROP elements contain the list of ServiceMessage properties in their

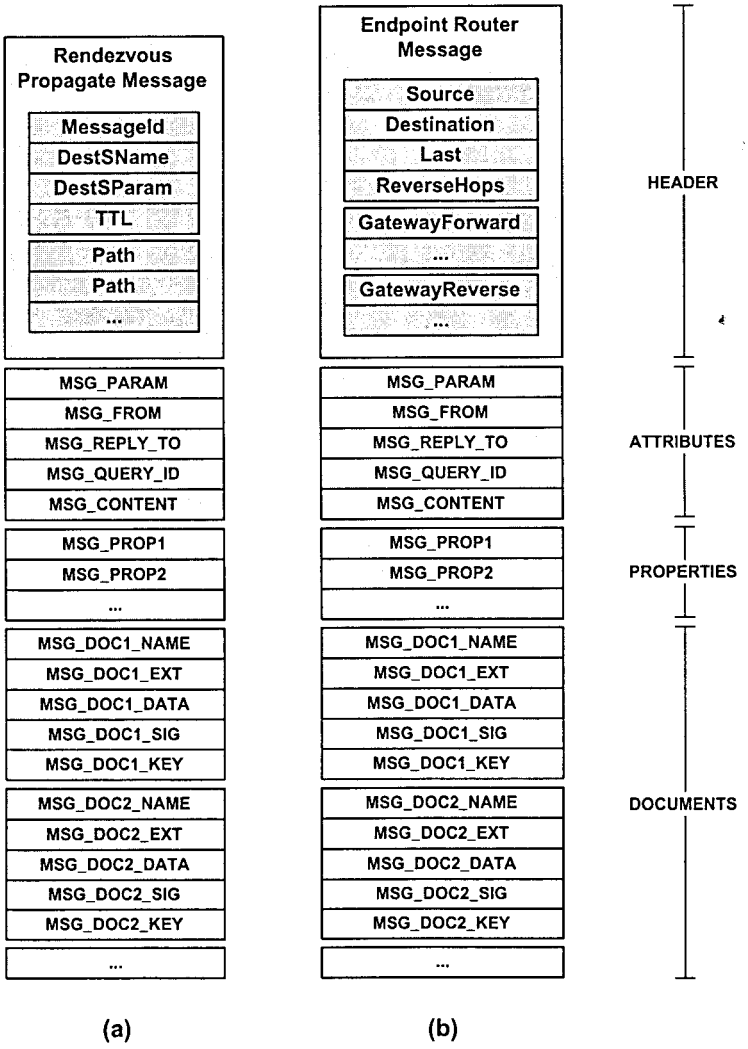


Figure 7.37: Multicast (a) and unicast (b) service messages

original form (property_name = property_value); these elements are numbered in order to simplify the parsing routines at the receiving end.

As for the ServiceMessage documents, sendMessage() actually loads the data files into the JXTA message. For each document in the ServiceMessage, sendMessage() creates at least three XML elements in the JXTA message: MSG_DOC_NAME, MSG_DOC_EXT, and MSG_DOC_DATA. The MSG_DOC_NAME element contains the document_name, and the MSG_DOC_EXT element contains just the file extension, which is extracted from the file_name. The MSG_DOC_DATA element contains the file's binary data encoded with Base64 [IETF, 1996]. Like in the MSG_PROP elements, the MSG_DOC elements are numbered in order to simplify the parsing routines at the receiving end. In some experiments, it was possible to exchange documents having a size of several megabytes.

It should be noted that including the complete file_name is not necessary, because once the file arrives at another system, that system may store it in a local file with another name. When a Service object at a remote peer receives a JXTA message, it creates a new ServiceMessage object and fills in its

attributes based on the JXTA message elements. When it comes to the document-related elements, the Service decodes the MSG_DOC_DATA element and saves its contents as a local file with an arbitrary name but with the extension specified in MSG_DOC_EXT. Then, the Service adds a new document to the ServiceMessage object in the form document_name = file_name, where document_name is the same as the MSG_DOC_NAME element, and file_name is the complete file path to the local file just created.

When a document is loaded into a JXTA message, the sendMessage() method retrieves the peer's private key and automatically signs the file's binary data with that private key. Then, sendMessage() creates two additional XML elements for the document: MSG_DOC_SIG and MSG_DOC_KEY, as shown in figure 7.37. The MSG_DOC_SIG element contains the document signature encoded with Base64, and the MSG_DOC_KEY contains the public key, also encoded with Base64, and which can be used to verify the document signature at the receiving end. When a Service receives a JXTA message, it checks if each document has the MSG_DOC_SIG and MSG_DOC_KEY elements; if it does, the Service decodes the MSG_DOC_SIG and MSG_DOC_KEY elements and automatically verifies the signature.

7.2.2 Integration with the Workflow Kernel

The Workflow Kernel is a reusable workflow enactment service because it defines a generic interface - the IAction interface - that should be implemented by any resource or external component that takes part in a workflow process. The Workflow Kernel also assumes that any component implementing the IAction interface will generate IEvent objects to signal the occurrence of relevant events, such as the completion of a previously requested task. Each IEvent object has some properties and documents associated with it, and every IEvent object must be delivered to the Workflow Kernel via the INotifySink interface. The problem is that IAction, IEvent, IProperty, IDocument and INotifySink are all COM-based interfaces, whereas the P2P integration infrastructure provides access to its functionality via a set of CORBA interfaces. Therefore, before the Action objects can be developed, there is a technology gap that must be overcome. This is the purpose of a module called PeerSvcs (from "Peer Services"), which translates COM method calls into CORBA method calls and vice versa.

7.2.2.1 The PeerSvcs module

The PeerSvcs module is a COM interface layer implemented as an in-process COM server (a DLL), just like in DAMASCOS (section 6.2.2.4 on page 320). Basically, the PeerSvcs module implements a set COM interfaces which have the same methods as the CORBA interfaces exposed by the integration infrastructure; these COM interfaces delegate methods calls to their CORBA counterparts. However, in addition to translating method calls from COM to CORBA, as the COM interface layer in DAMASCOS does, the PeerSvcs module must be able to translate method calls from CORBA back to COM since one of the CORBA interfaces - the ServiceListener interface - is actually a callback interface. Fortunately, the ServiceListener interface has only one method, so there is only one CORBA method that the PeerSvcs module must translate into a COM method call.

Figure 7.38 shows the five COM interfaces in the PeerSvcs module, together with their relationships to CORBA interfaces exposed by the integration infrastructure. The INameService interface allows a COM client to retrieve a reference to a ServiceManager object registered in the CORBA Naming Service; for this purpose, the COM client must provide the group name as input parameter to the getServiceManager() method. Optionally, the COM client may invoke the listServiceManagers()

method in order to find out which ServiceManager objects are available. The INameService object does not return a ServiceManager reference to the COM client; instead, it creates a new COM object that implements the IServiceManager interface, which is a COM wrapper to the ServiceManager CORBA interface.

When the COM client invokes the IServiceManager interface in order to create Service objects, the object implementing IServiceManager delegates the call to the ServiceManager object, which creates either a UnicastService object or a MulticastService object, according to the original method call. In the COM side, both of these CORBA interfaces are represented by a single IService interface. The IService COM interface contains all methods defined in the Service CORBA interface, plus the two specialized methods that the UnicastService and MulticastService interfaces contain: listEndpointAddresses() and getServiceAdvertisement(). Regardless of whether a COM client invokes createUnicastRequest() or createMulticastRequest(), it receives an IService pointer as result.

If the COM client invokes the listEndpointAddresses() method, which comes originally from the UnicastService CORBA interface, on an IService interface that actually represents a MulticastService, the COM object implementing IService does nothing. The same happens when a COM client invokes the getServiceAdvertisement() method, which comes originally from the MulticastService CORBA interface, on an IService interface that represents a UnicastService CORBA object.

When a COM client invokes the IService interface in order to create a request or reply message, the object implementing IService delegates the call to the CORBA interface that it is encapsulating. When the CORBA method returns the newly created ServiceMessage object, IService creates a new COM object that implements the IServiceMessage COM interface, it associates this object with the ServiceMessage CORBA object, and it returns the IServiceMessage COM object.

The IServiceManager, the IService and the IServiceMessage interfaces are implemented by COM objects that provide access to an equivalent CORBA interface. Thus, the PeerSvcs module must have some way to associate COM references with CORBA references. A possible approach is to include,

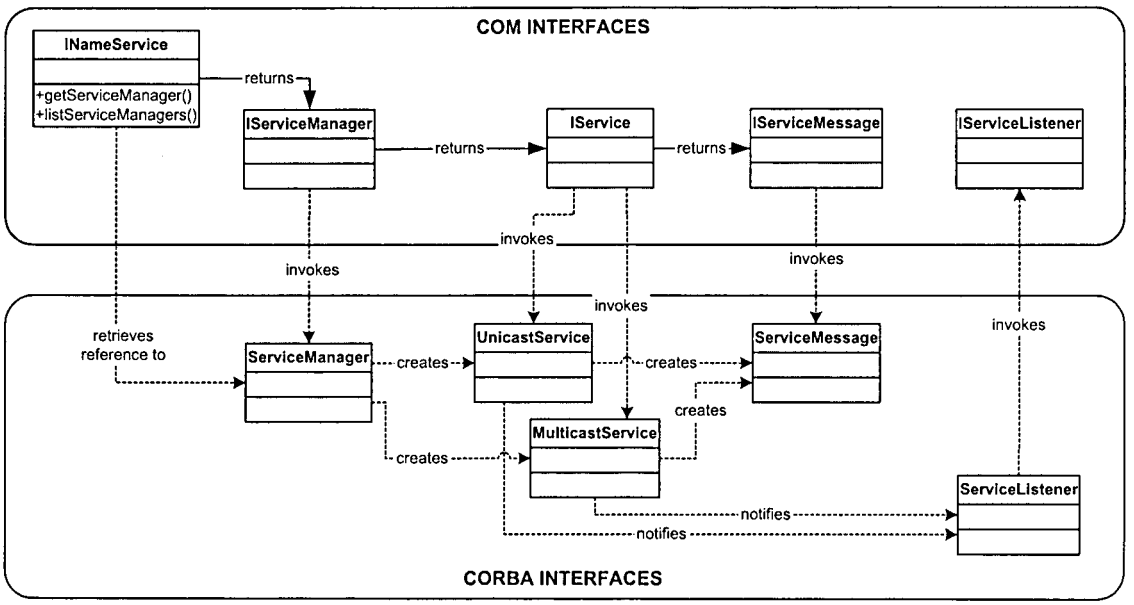


Figure 7.38: Relationships between COM and CORBA interfaces

in the implementation class of every COM object, an attribute that stores a reference to the CORBA object being encapsulated. The PeerSvcs module employs another approach: it defines a globally accessible map that associates IUnknown interface pointers with CORBA::Object references.

Typically, the implementation of a COM method starts by looking up the CORBA::Object reference associated with the IUnknown pointer to the COM interface that contains that method. Then, the COM method casts the CORBA::Object reference into its particular type by invoking a standard CORBA operation called _narrow(), and it invokes the corresponding method on that CORBA object. If the CORBA method returns a reference to another CORBA object, the COM method creates a new COM object that encapsulates that CORBA object and inserts a new entry in the global map, associating the COM object's IUnknown pointer with the new CORBA object reference.

7.2.2.2 The IServiceListener interface

In general, the PeerSvcs module implements a set of COM interfaces that delegate calls to the CORBA interfaces implemented by the integration infrastructure. Such is the case of the IServiceManager, the IService and the IServiceMessage COM interfaces, which delegate calls to the ServiceManager, UnicastService, MulticastService and ServiceMessage CORBA interfaces. The PeerSvcs module includes an additional COM interface - the IServiceListener interface - that encapsulates the ServiceListener CORBA interface. The difference between IServiceListener and the remaining COM interfaces is that IServiceListener encapsulates a CORBA interface that is actually implemented by the PeerSvcs module itself, not by the integration infrastructure.

As explained in section 7.2.1.2, the ServiceListener interface is a callback interface to be implemented by any external component or application in order to be notified of received messages. Therefore, the PeerSvcs must implement ServiceListener; otherwise, it will not be able to receive any incoming messages. The PeerSvcs module defines ServiceListenerImpl, which is a C++ class that implements the ServiceListener interface using the TAO ORB. In addition, it defines IServiceListener, which is a COM interface that encapsulates the ServiceListener interface.

The PeerSvcs module can subscribe to messages received by a given Service according to the procedure illustrated in figure 7.39. First, the COM client creates a COM object implementing the IServiceListener interface (step 1). This COM object automatically creates an instance of the ServiceListenerImpl class, which implements the ServiceListener CORBA interface (step 2). Then, the COM client invokes the IService's addServiceListener() method, passing the IServiceListener interface pointer as input parameter (step 3). Using the global map, the addServiceListener() COM method will retrieve two CORBA::Object references: one for the Service object associated with this IService interface, and the other for the ServiceListener object associated with the supplied IServiceListener interface pointer. Then, the addServiceListener() COM method will invoke the addServiceListener() CORBA method, passing the ServiceListener object reference as input parameter (step 4). The Service object adds the supplied ServiceListener reference to the list of ServiceListener objects that it has to notify. As soon as the Service object receives a new JXTA message, it notifies the ServiceListenerImpl object by invoking the receiveServiceMessage() method (step 5), passing a newly created ServiceMessage object as input parameter.

Once a ServiceMessage reaches the PeerSvcs module, there must be some way for the PeerSvcs module to notify this occurrence to external COM components. Clearly, this requirement suggests the use of a callback interface that COM components should implement in order to receive message notifications from the PeerSvcs module. Instead of introducing a new interface, the natural solution is to use the IServiceListener message itself. Additionally, external COM components should make

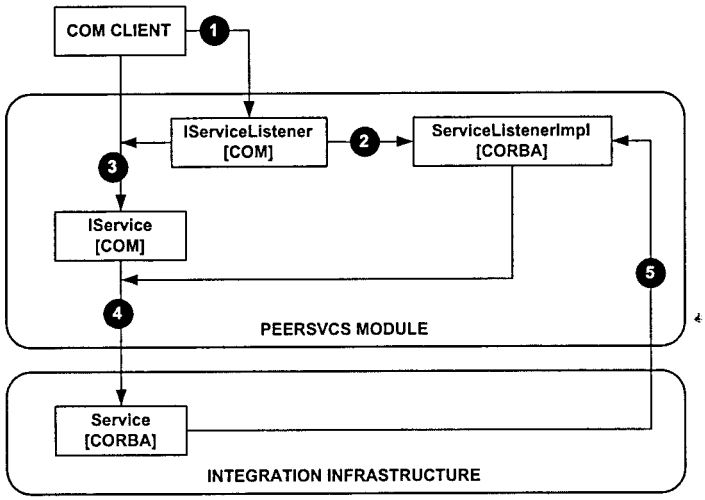


Figure 7.39: Subscribing to messages via the PeerSvcs module

use of connection points in order to subscribe and unsubscribe to message notifications. Therefore, the COM object that implements the IServiceListener interface within the PeerSvcs module must also implement a connection point for that same interface. Whenever its receiveServiceMessage() is invoked, that COM object delegates the method call to external COM objects that also implement the IServiceListener interface.

The complete chain of message notification is illustrated in figure 7.40:

- At the bottom layer (layer 1) there is the integration infrastructure, which is able to produce a new ServiceMessage object for each JXTA message received via the RendezVousService or via the EndpointService. Each Service object is able to notify this occurrence to any CORBA object that implements the ServiceListener CORBA interface and that has subscribed to incoming messages by invoking the Service's addServiceListener() method.

- The PeerSvcs module (layer 2) implements the ServiceListener interface by means of a C++ class called ServiceListenerImpl. Within the ServiceListenerImpl class, the receiveServiceMessage() method creates two COM objects that encapsulate the supplied input parameters: one COM object encapsulates the Service reference, while another COM object encapsulates the ServiceMessage reference. The ServiceListenerImpl class then invokes the receiveServiceMessage() method of an IServiceListener COM interface; the COM object that implements this interface must have been previously created by an external COM client. When ServiceListenerImpl invokes the IServiceListener's receiveServiceMessage() method, it supplies two input parameters: an IService interface pointer and an IServiceMessage interface pointer.

- The PeerSvcs module implements a connection point for the IServiceListener interface, which allows external COM clients (layer 3) to receive message notifications. The PeerSvcs module delegates the receiveServiceMessage() method calls to the COM clients' IServiceListener interfaces. The COM clients are Action objects, which use the PeerSvcs module in order to exchange messages with remote peers over the integration infrastructure. Typically, an Action

object will be interested in receiving a reply to a previously sent request; therefore, it must implement the IServiceListener interface.

- Each received message may or may not represent a relevant workflow event that should be notified to the Workflow Kernel (layer 4). If a message signals the completion of a previously requested task, the Workflow Kernel must be notified of this occurrence in order to proceed with the execution of the following actions. Each Action object is expected to notify the Workflow Kernel by creating an Event object and delivering this event via the INotifySink interface to its associated Place object and to any additional objects that have explicitly subscribed to the events generated by this Action.

From the experience of implementing the PRONEGI's XFlow application (section 6.1.3.7 on page 303) it was anticipated that, for such a sequence of callback calls to work properly, the CORBA and COM threading models would have to be conciliated at some point. As a matter of fact, it was found that this conciliation involves two issues:

1. The COM library must be initialized for CORBA threads. When the Service object receives a JXTA message it notifies the ServiceListenerImpl object by invoking its receiveServiceMessage() method, but this method call is made from within a CORBA thread. Since the ServiceListenerImpl's receiveServiceMessage() method creates and invokes COM objects, the COM run-time libraries must have been previously initialized for this CORBA thread. This has been achieved by initializing the COM library in the same thread that initializes the ORB.

2. Interface pointers must be marshaled between the WfAction module and the WfKernel module. The PeerSvcs module has been implemented as an in-process COM server, so its objects live in the Single-Threaded Apartment (STA) of the WfAction module. But when a COM object from the WfAction module (layer 3 in figure 7.40) invokes a COM object from the WfKernel module (layer 4), this method call crosses over into the Workflow Kernel's STA. Therefore, the WfAction module must marshal any pointers to Workflow Kernel interfaces before invoking

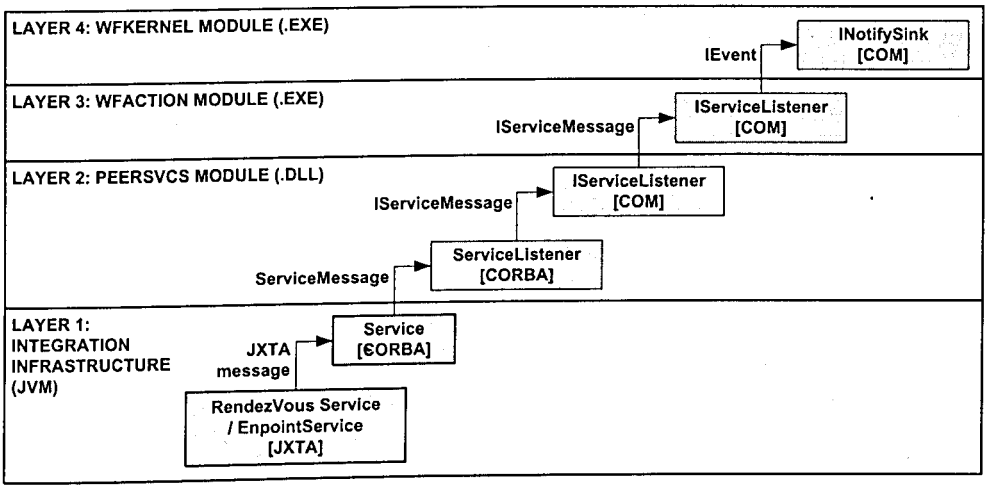


Figure 7.40: Message notification up to the Workflow Kernel

them. This is achieved by means of the CoMarshalInterface() and CoUnMarshalInterface() COM functions[134].

7.2.2.3 Workflow actions and service messages

As explained earlier in section 7.1.3.1 on page 362, an action is an event-delimited amount of work that is performed by some resource. An action is initiated upon request by the Workflow Kernel, which expects the resource to return an output event as soon as it completes the action. At the P2P integration infrastructure level, the action starts when a request message is sent to a remote peer and it may end when a reply message is received from that peer. In an inter-enterprise scenario, sending a request and receiving a reply is the basic behavior that allows more complex interactions to be built. Ultimately, it is the ability to send a request and to receive a reply that enables enterprises to integrate their business processes. For example, RosettaNet PIPs make use of the request/reply concept in order to integrate processes running at different enterprises, as suggested in figure 5.24 on page 202.

This explains why the proposed integration infrastructure makes a sharp distinction between messages that represent requests and those that represent replies. As described in section 7.2.1.3, a ServiceMessage object may represent a request or a reply; either one of the two methods isRequest() and isReply() says whether the message represents a request or a reply. Although requests and replies are represented by the same kind of object, they have different semantics. When a Service sends a request, it automatically subscribes to endpoint messages in order to be able to receive any reply; regarding replies, the Service just sends the message via the EndpointService.

There are two complementary perspectives for any given P2P service: one is that of a service client, which sends a request and waits for some reply; the other is that of a service provider, which waits for an incoming request, handles the request, and produces a reply. (The difference between unicast services and multicast services is only that a unicast service has a single service provider, whereas a multicast service may have many simultaneous service providers.) Therefore, the service provider is assumed to be listening for requests when the service client sends the request, and the service client is assumed to be listening for replies when the service provider sends a reply. Between the request and the reply, the service provider is handling the request while the service client is waiting for the service provider to respond.

Using the Workflow Kernel, this behavior can be modeled as shown in figure 7.41. This modeling approach requires three kinds of actions: one to send requests, one to receive requests, and another to send replies. The first action supports the role of the service client, whereas the second and third actions support the role of the service provider. Each of these actions can be implemented as a single Action object, which relies on the PeerSvcs module to send and/or receive messages over the integration infrastructure. It should be noted that there is no need to have an Action whose purpose is to wait for a reply, because a Service automatically subscribes to replies when it sends a request. However, it is necessary to distinguish between Actions that deal with unicast services and those that deal with multicast services, since these services have different ways to send requests.

Furthermore, it should be possible to support multi-task activities, as discussed in section 6.3.4 on page 339. This kind of activity requires an Action that is able to send a request to a multiple service providers. This is different from sending a multicast request: in a multicast request, the service client does not know which service providers will actually receive the request, whereas in a multi-task activity the service client knows exactly which service providers the request should be sent to. After

[134]Some information about the use of these COM functions, as well as sample code, can be found in the Microsoft Knowledge Base (http://msdn.microsoft.com/), article Q206076.

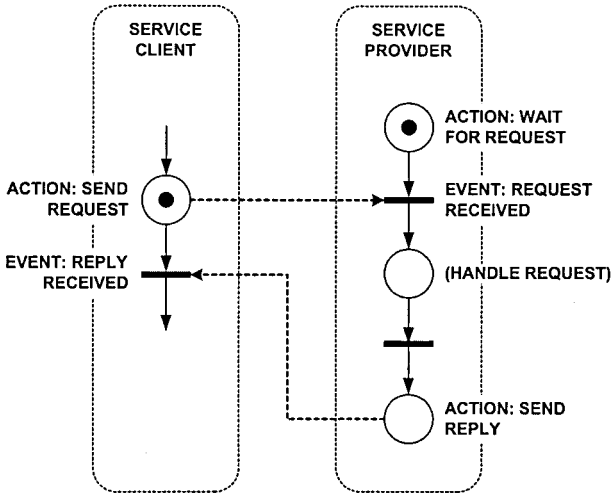


Figure 7.41: Modeling request/reply behavior with the Workflow Kernel

having sent the request, the service client may expect a number of replies that is equal to or less than the number of service providers the request has been sent to.

Altogether, six types of Actions have been defined: *Run Multicast Service* (RMS), *Run Unicast Service* (RUS), *Send Multicast Request* (SMR), *Send Unicast Request* (SUR), *Send Multiple Unicast Requests* (SMUR), and *Send Service Reply* (SSR):

- RMS and RUS are Actions that wait for incoming requests: RMS creates a MulticastService object and invokes its startService() method, which starts listening for incoming messages on an input pipe; RUS creates a UnicastService object and invokes its startService() method, which subscribes to incoming messages from the EndpointService.

- SMR and SUR are Actions that send requests and wait for replies: SMR creates a MulticastService object and invokes the createRequest() method, which creates a request message, followed by the sendMessage() method, which sends the request message through a propagate pipe; SUR creates a UnicastService object and invokes the createRequest() method, which creates a request message, followed by the sendMessage() method, which sends the request message to a specified endpoint address.

- SMUR is an Action that sends a request to multiple endpoint addresses. SMUR creates a UnicastService object and invokes the createRequest() method, which creates a ServiceMessage object. For each of the endpoint addresses that the message should be sent to, SMUR sets the ServiceMessage's DestinationAddress attribute and invokes the UnicastService's sendMessage() method, passing the ServiceMessage as input parameter.

- SSR is an Action that sends a reply to a previous request: first, SSR obtains a reference to a MulticastService or UnicastService with a given name, and then it invokes the createReply() method, which creates a reply message, followed by the sendMessage() method, which sends the reply message to a specified endpoint address.

7.2.2.4 Service messages and JXTA messages

Each and every Action has a set of properties, which hold input data for the Action. For example, all Actions must be given the name of the service they act upon, and some Actions need specific parameters such as the destination endpoint address (SUR and SSR). These are mandatory properties, which have a direct impact on how an Action operates; these properties are workflow relevant data. In addition to these mandatory properties, all Actions may be given customized properties which contain information that is relevant to perform tasks; these properties are workflow application data. For example, SMR could be given an additional property that specifies the period of time during which the request is valid; this property does not have an influence on how the Action operates, but it should be sent together with the request.

Each Action has also a set of documents, which refer to data files that are relevant within a certain process or within a certain process instance. Each document has a name and a location: the name should be defined prior to process execution, whereas the actual location of the file may be given a pre-defined value or it may be set during process execution, as described in section 7.1.4.7 on page 378. Whenever the location of a document is left empty, this represents one of two possibilities: either the document effectively represents another document produced by an earlier action, or it represents an output document of the current action. Even if a document refers to an existing file, during its operation an action may change the file, replace the file or produce additional output files. The same applies to customized properties, whose values can be changed during the execution of the action, and afterwards returned as output properties.

Actions return output properties and output documents by means of Events that they deliver to the Workflow Kernel via the INotifySink interface. When an Action is notified of a received message, it creates an Event object. For RMS and RUS, the generated Event represents a received request, whereas for SMR and SUR the Event represents a received reply. The Action adds some pre-defined properties to the Event such as the endpoint address of the sender; in addition, it loads the Event with the customized properties and documents brought by the message. Figure 7.42 illustrates how an Action fits its own properties and documents into a JXTA message, and how it extracts properties and documents from a JXTA message into an Event object.

7.2.2.5 Pre-defined vs. customized properties

Figure 7.43 shows the pre-defined properties for each Action, as well as the Events that each Action generates. All Actions have a PeerGroup property which effectively identifies the ServiceManager object they should use, since each ServiceManager object is registered in the Naming Service with the name of the peer group. The ServiceName property specifies the service that the Action must use to interact with remote peers. For both RMS and RUS, PeerGroup and ServiceName are the only pre-defined properties. Each of these Actions generates an EVT_REQUEST Event with two pre-defined properties: the RequestPeer property contains the endpoint address of the original sender and the QueryID property contains the value of the QueryID message attribute.

In the SMUR Action, the TargetPeers property specifies all the endpoint addresses the message should be sent to, separated by semicolons (";"). The MinReplies property specifies the minimum number of replies that should be received before generating an EVT_RESULTS Event, which contains the sum of all properties and documents collected from replies. The EVT_RESULTS Event has a SourcePeers property, which contains the endpoint addresses from all replies, again separated by semicolons. An EVT_TIMEOUT Event will be generated if SMUR does not receive the minimum

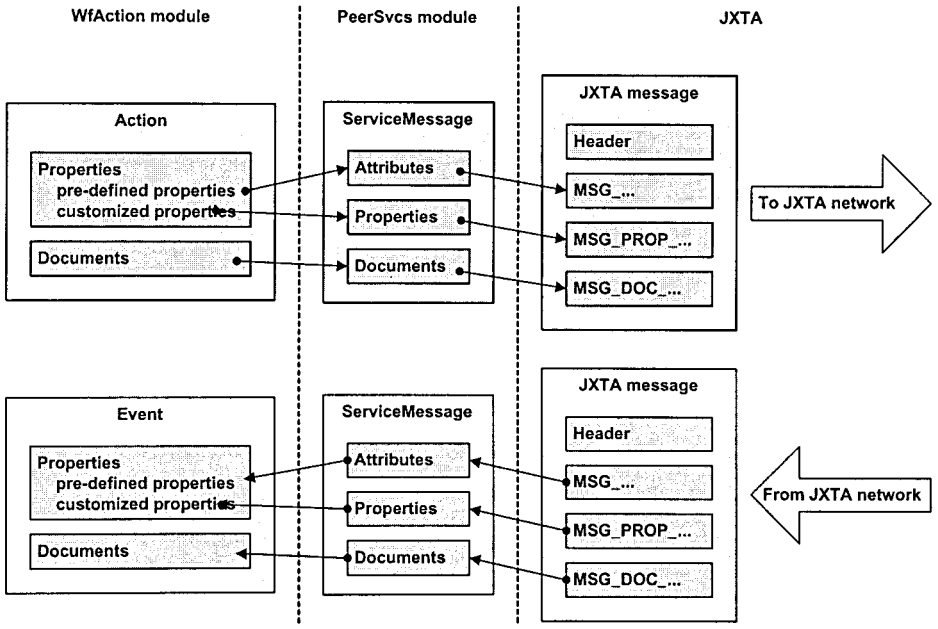


Figure 7.42: Properties and documents across software layers

number of replies within a number of seconds specified by the TimeOut property; in this case, the SourcePeers property contains the endpoint addresses from the replies received so far.

The SMR Action has one TargetRole property, which may specify the name of a business role that recipients should be able to perform; if this property is given a value, then only the recipients that are able to play the specified role should reply to the request. The TimeOut property specifies the maximum amount of time the Action will wait for some reply, the TimeWait property specifies the maximum amount of time the Action will spend collecting replies, and the MaxHits property specifies the maximum number of replies that should be collected. If no reply is received within TimeOut seconds, SMR generates an EVT_TIMEOUT Event with an empty SourcePeers property; otherwise, SMR will generate an EVT_RESULTS Event either as soon as MaxHit replies have been collected or as soon as TimeWait seconds have elapsed since the Action has started. The SourcePeers property in the EVT_RESULTS Event contains the endpoint addresses from all replies.

In SUR, the TargetPeer property specifies the endpoint address that the request should be sent to, and the TaskDescription property contains any information that is relevant to the remote peer which receives the request. When a SUR Action receives a reply, it creates either an EVT_AFFIRMATIVE Event or an EVT_NEGATIVE Event, depending on the reply's Content attribute: if this attribute contains any one of the words "affirmative", "positive", "yes", or "true", then SUR generates an EVT_AFFIRMATIVE Event, otherwise it generates an EVT_NEGATIVE Event. The ResponseLabel property contains the original value of the reply's Content attribute. The ReplyPeer property contains the endpoint address of the peer that sent the reply.

In SSR, the TargetPeer property specifies the endpoint address that the reply should be sent to, and the QueryID property specifies the value for the QueryID message attribute. The ResponseLabel property is a descriptive field, which is used mainly to express an affirmative or negative reply to a previous request. The ResponseLabel property is only used with unicast services: an affirmative

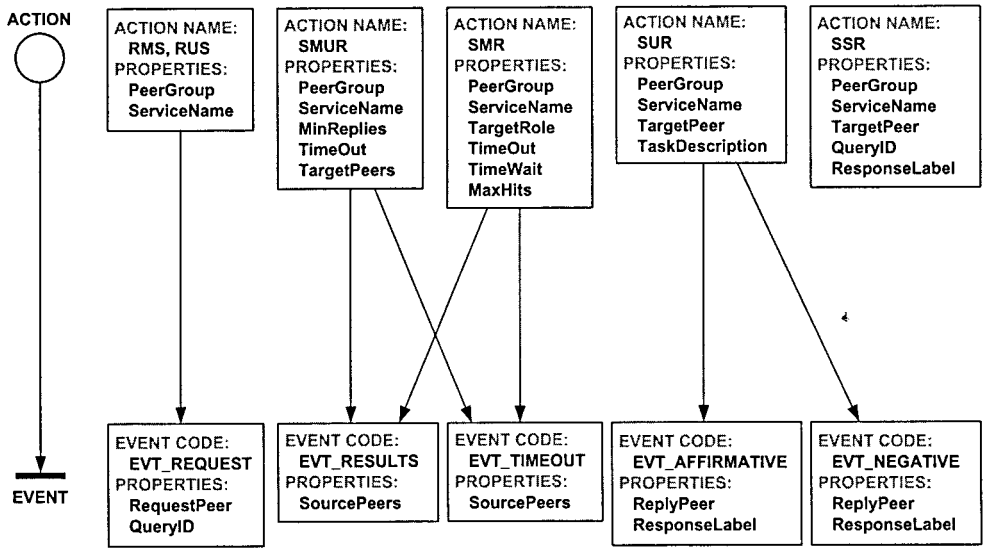


Figure 7.43: Pre-defined properties for Actions and Events

reply means that either the task has been completed according to the original request, or that the input data in the original request was accepted; a negative reply means that either the task could not be successfully completed or that the input data was not accepted.

The output information coming with an Event can be passed as input information to the following Actions. For example, the value of the RequestPeer property in an EVT_REQUEST Event can be used as the value for the TargetPeer property in a subsequent SSR Action. In general, the value of any output property (i.e. an Event's property) can be assigned to an input property (i.e. an Action's property) if the evaluate attribute of the input property contains the name of the output property. During process execution, the Workflow Kernel will assign the value of the output property to the input property as explained in section 7.1.4.7 on page 378. Regarding output documents, these documents will be passed as input documents to the following Actions if and only if those Actions have a Document with the same name, as explained in the same section.

When properties and documents other than the pre-defined properties are added to an Action, they are automatically added to the Action's output Events as well. These customized information items may represent either input data for the Action or output data coming from the Action. In principle, the properties and documents that represent input data should be added to Actions, not to Events, and the properties and documents that represent output data should be added to Events, not to Actions. However, there are good reasons to add any customized properties and documents to Actions and Events simultaneously. On one hand, there must be some way to specify beforehand which outputs are expected from an Action, and this is indeed a kind of input information that must be provided to the Action; hence, the chosen approach is to add output properties and documents to the Action and to leave their values and locations unspecified. On the other hand, the input properties and documents that are supplied to an Action may have to be supplied to subsequent Actions, so they can be made available as outputs by adding them to Events. Finally, having an information item both as an input and as an output for an Action allows this item to be updated and then sent back to the workflow enactment service.

7.2.2.6 The QueryID property

Both RMS and RUS generate an EVT_REQUEST Event having a pre-defined QueryID property, which contains the QueryID attribute value of the received request message. When replying to the request, the reply's ServiceMessage object must contain this same value for its QueryID attribute. This explains why SSR has a QueryID property; SSR must know the QueryID value of the original request in order to send the reply. When SMR or SUR generate the request message, these Actions set the QueryID attribute's value as the Token color passed as the input parameter to their Start() method. When SMR or SUR generate an Event, they set the Event's color to that same Token color, so that any following Transition is fired for that color.

The way the QueryID attribute goes from the service client to service provider and then back to the service client is illustrated in figure 7.44. First, it is assumed that service provider has started a RMS Action, which waits for an incoming request. RMS starts when its associated Place receives a new Token (step 1); the Action's Start() method is invoked, passing the Token's color as input parameter (step 2). On the service client side, the Place associated with a SMR Action receives a new Token (step 3), and the Action's Start() method is invoked, passing the Token's color as input parameter (step 4). SMR sends a JXTA message using the PipeService and having a MSG_QUERY_ID element with value "color1" (step 5).

When the RMS Action at the service provider receives this message, it creates a new EVT_REQUEST Event with a QueryID property whose value is "color1" (step 6). This Event triggers Transition t1, which removes the Token from Place p1 and inserts a new Token into Place p2 (step 7). Place p2 will have some Action associated with it, whose purpose is to handle the received request; this Action, which is not shown in figure 7.44, could invoke a local resource using SUR, for example. After the request has been handled, Transition t2 fires, removing the Token from Place p2 and inserting a new Token into Place p3 (step 8). Place p3 has an associated SSR Action, so it

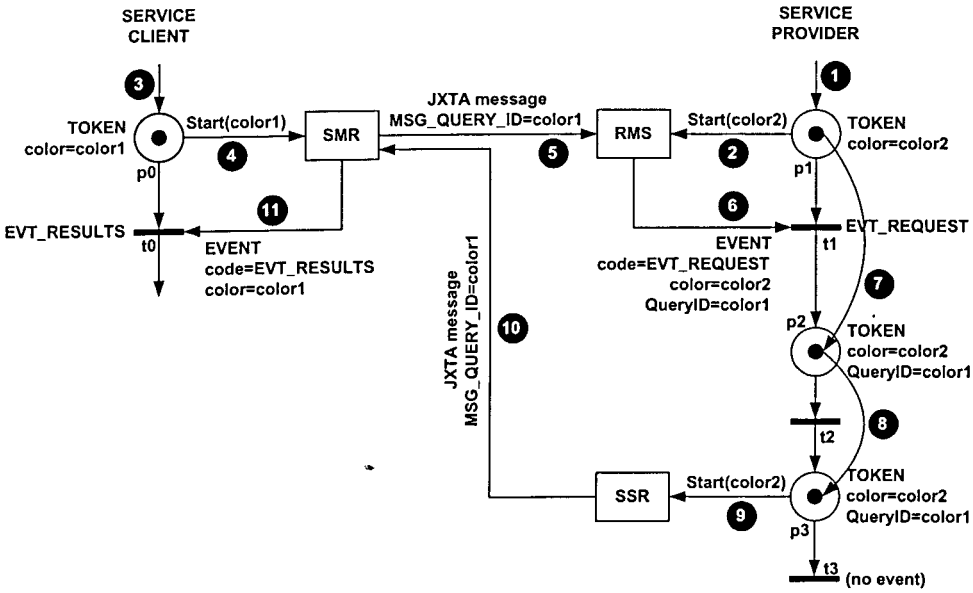


Figure 7.44: Pre-defined properties in Actions and Events

updates this Action's properties (which include QueryID) and invokes the Start() method, passing the Token's color as input parameter (step 9). SSR creates a reply message and uses the QueryID property to set the value of the MSG_QUERY_ID message element.

The SMR Action at the client side receives this reply message (step 10) and, since it has invoked a MulticastService, it may receive replies from other peers too. As soon as the maximum number of replies (MaxHits) is reached or as soon as TimeWait seconds elapse, the SMR Action creates an EVT_RESULTS Event (step 11). The Event's color is set as the value of the MSG_QUERY_ID element in the received message. The Event triggers Transition tA, and process execution proceeds to the following Actions.

7.3 Proposed integration architecture

The combination of the Workflow Kernel with the P2P integration infrastructure is the proposed solution to the challenge of supporting the engineering of business networks. The P2P integration infrastructure is a common and decentralized messaging platform that provides the necessary services for peers to discover and interact with each other. In a business network these peers represent individual enterprises, and peer groups represent marketplaces, i.e., virtual trading zones where enterprises can meet and conduct business with each other. The access to a peer group may be restricted to specific members, or it may be open to any enterprise which connects to the network. In addition, any enterprise may create its own group and invite other enterprises to join the group. Within a peer group, enterprises can establish temporary or permanent business relationships with each other and, at any time, they may reorganize themselves according to a different configuration, thus creating a new business network.

The combination of the Workflow Kernel with the P2P integration infrastructure provides enterprises with the ability to discover each other, and with the ability to integrate business processes with one another. This combination results in an integration architecture that supports both intra- and inter-enterprise business processes, and both operational and engineering business processes. The internal processes of an enterprise are regarded as sub-processes within inter-enterprise processes, and the operational processes are regarded as sub-processes within engineering processes. This work focuses on supporting inter-enterprise processes and engineering processes, and expects that the devised approach can cope with internal processes and operational processes as well. This way, the combination of the Workflow Kernel with the P2P integration infrastructure results in a generalized integration architecture that can support both intra- and inter-enterprise processes.

The proposed integration architecture is illustrated in figure 7.45 which, for simplicity, shows only two enterprises. Each enterprise runs its own instance of the Workflow Kernel, and the integration infrastructure grows as more enterprises connect to the P2P network. Using the Workflow Kernel, enterprises can define their own engineering processes; they can also set up their operational processes, although these may ultimately depend on the contracts that enterprises establish with each other. In the same way, figure 7.45 displays an engineering process that is basically equivalent to the B2B trading life cycle presented in section 4.4.3 on page 161, although enterprises are free to define other kinds of engineering processes. This kind of flexibility in defining business processes is precisely one of the advantages of having a workflow management system; whereas existing workflow management systems focus on operational processes, this work generalizes workflow management to engineering processes.

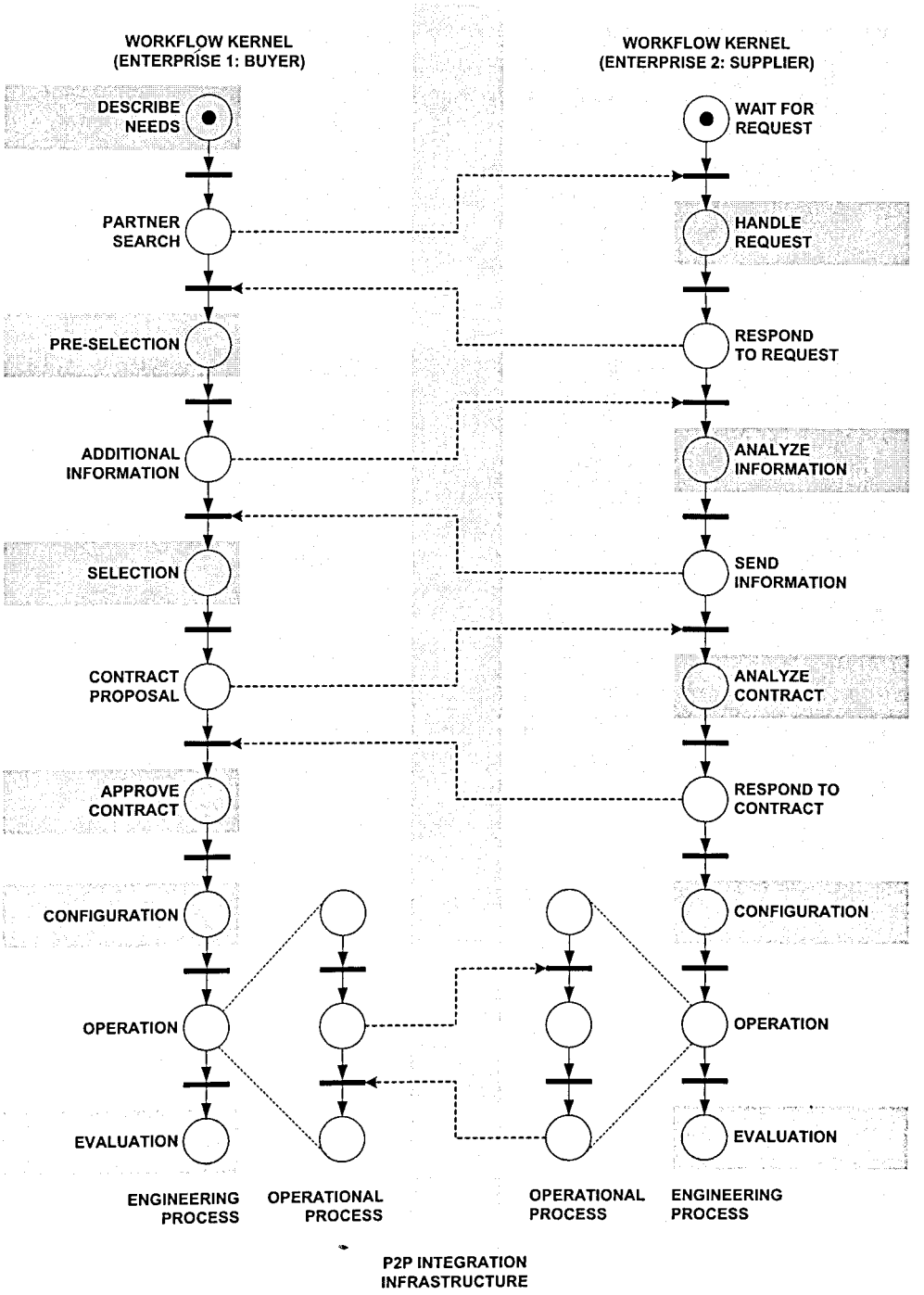


Figure 7.45: Integration architecture for inter-enterprise business processes

7.3.1 Service client/provider roles

Figure 7.45 suggests that there are two sides for an engineering process: one is that of buyers and the other one is that of suppliers. But figure 7.45 also suggests that buyers behave mostly as a service clients, i.e. clients of P2P services, because they send requests and wait for replies; conversely, suppliers behave mostly as service providers, since they receive requests and produce replies. However, this does not prevent these client/provider roles to be inverted during process execution.

The existence of these two roles is a consequence of the fact that whenever one peer sends a request, it expects other peers to be listening to requests, and whenever one peer sends a reply it expects that the peer that sent the original request is waiting for a reply. In terms of the Service interface discussed in section 7.2.1.2, it is assumed that the provider has already invoked the startService() method by the time the client invokes sendMessage() to send the request, and it is assumed that the client is still listening for replies when the provider invokes sendMessage() to send the reply. If this is not the case then process execution gets stuck, because those events do not reach the Workflow Kernel, hence the corresponding transitions are not fired.

Therefore, the process running at the service provider should be consistent with that running at the service client, so that each message that is sent is indeed being expected at the receiving end. If necessary, the definition of the engineering process could be published as an advertisement in the JXTA network.

7.3.2 External support services

The integration architecture depicted in figure 7.45 requires two different kinds of P2P services: multicast (one-to-many) services that allow a peer to discover new business partners, and unicast (one-to-one) services that allow peers to attain exchanges with those partners. These two kinds of services - multicast and unicast - correspond to the two basic networking mechanisms available in most P2P platforms; this is especially evident in the case of Gnutella, which defines a broadcast mechanism to search for files and a direct connection mechanism to transfer files between peers. The P2P integration infrastructure that was built on top of JXTA is a general-purpose service platform that allows enterprises to create and make use of multicast and unicast services in order to exchange messages with each other. To support the engineering process shown in figure 7.45, one multicast service and three unicast services have been defined. These services support the first four phases of the B2B trading life cycle described in section 4.4.3:

- *Trading Partner Search Service* (TPSS) - this multicast service is intended to support partner search. TPSS can make use of TOs and TRs, as defined by the IRC Network (appendix A), to describe selling offers and purchase needs, respectively. Whenever a buyer creates a new TR, it sends this TR through TPSS and expects suppliers to return candidate TOs. There is no central storage of TOs and TRs, but enterprises are free to cache previously received documents. The TPSS service client must send a TR within a document called TechnologyRequest, and the TPSS service provider must send a TO within a document called TechnologyOffer.

- *Trading Partner Information Service* (TPIS) - this unicast service allows an enterprise to retrieve more information about previously received TOs (or TRs). At first, the enterprise will want to know who the potential supplier or buyer is (since TOs and TRs are anonymous), so TPIS can be used to exchange company profile information. The company profile may also include further details about the product being offered or sought. The TPIS service client must

send its company profile within a document called BuyerInfo, and the TPIS service provider must send its company profile within a document called SupplierInfo.

- *Trading Partner Agreement Service* (TPAS) - the purpose of this unicast service is to support the contracting phase, during which buyer and supplier may exchange several contract proposals until they reach an agreement. As discussed in section 6.3.7 on page 342, a Trading Partner Agreement (TPA) should describe the network-level process, i.e., the sequence of message exchange steps that will take place between enterprises during the fulfillment phase. For this purpose, enterprises could possibly use the tpaML language, whose <Request> and <Response> elements (section 5.2.7.2 on page 206) can easily be mapped onto request messages and reply messages, respectively. Once they reach an agreement, enterprises can configure their operational processes according to the sequence of message exchange steps specified in the TPA. The TPAS service client must send a TPA proposal within a document called ContractProposal, and the TPAS service provider must send a response within a document called ContractResponse.

- *Trading Partner Exchange Service* (TPES) - this unicast service supports all exchanges during the fulfillment phase. TPES is the only service that operational processes use in order to exchange messages with external business partners. Enterprises are encouraged either to make use of document formats that have been specified by existing B2B frameworks, or to adapt these documents according to their own requirements. The TPES service client and the TPES service provider should refer to the TPA for information on the documents to exchange during the fulfillment phase.

7.3.3 Internal support services

Figure 7.45 shows that there are two different kinds of actions: there are actions which represent message exchanges with an external business partner (e.g. Contract Proposal), and there are actions which represent internal activities that must be performed locally within the enterprise (e.g. Contract Approval). These internal activities, which are marked with shaded areas, must be assigned to internal resources. For each activity, there may be a single resource with the required competence, or there may be several resources which are able to perform it. In the first case, the activity must be assigned to the one and only resource that should perform it (user assignment), whereas in the second case the activity may be assigned to one of several resources who can play the same role (role assignment). These are basically the two resource allocation models discussed in section 6.3.1 on page 334.

These resource allocation models can be implemented with unicast and multicast services: unicast services can be used to exchange messages with a specific resource, and multicast services can be used to inquire several resources simultaneously. Therefore, resource integration within the enterprise can be achieved with the same kind of P2P integration infrastructure. This does not mean that enterprise resources are connected to the same infrastructure used to communicate with other enterprises. Instead, there are two infrastructures: one for inter-enterprise integration and another for intra-enterprise integration. The link between the inter-enterprise environment and the intra-enterprise environment are the processes running at the Workflow Kernel, as suggested in figure 7.46.

To support the interaction with internal resources, one multicast service and one unicast service have been defined:

- *Local Resource Assignment Service* (LRAS) - this multicast service is intended to support role assignment of workflow activities to local resources. This kind of assignment comprises two

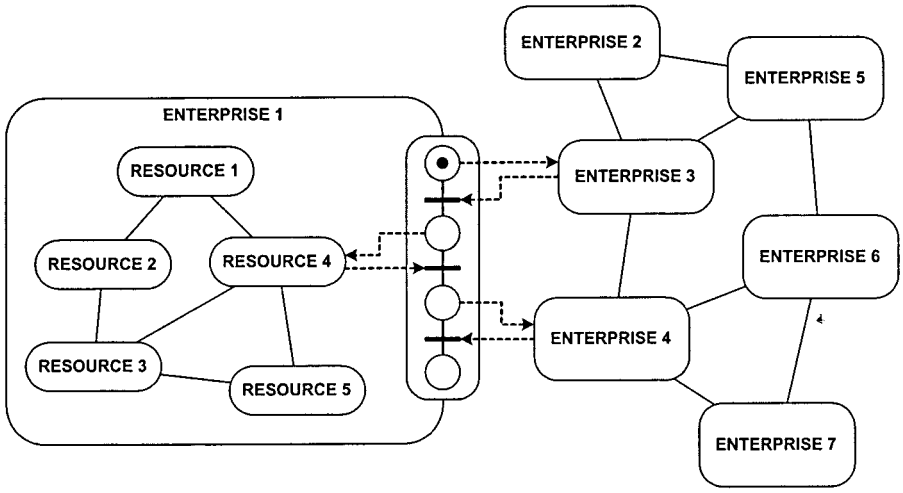


Figure 7.46: Linking inter-enterprise processes to intra-enterprise processes

phases: in the first phase, a multicast request is sent to all resources within a peer group, from which only those resources that can play the specified role are expected to reply; in the second phase, a unicast request is sent to the first resource that is available to perform the activity. LRAS supports just the first phase, i.e., it can collect replies to the multicast request, but another service must be used to interact with the ultimately assigned resource.

- *Local Resource Task Service* (LRTS) - this unicast service supports all exchanges during activity execution, from initial request to activity completion. LRTS can only be used when the assigned resource is already known; this resource may be pre-defined in the process model (user assignment) or it may be found at run-time by means of a multicast service such as LRAS (role assignment). LRTS supports the exchange of any properties and documents that represent either input data or output data for a given activity. In the process depicted in figure 7.45, all internal activities are performed as human tasks.

7.3.4 The engineering process

The P2P support services can be invoked from within the Workflow Kernel by means of the Actions described in section 7.2.2.3. These Actions - RMS, RUS, SMR, SUR, SMUR, and SSR - support the behavior of both service clients and service providers. In general, service clients will make use of SMR, SUR and SMUR, whereas service providers will make use of RMS, RUS and SSR. This set of six Actions can be used to invoke both external and internal P2P support services. However, these kinds of services are available in different peer groups, if not in distinct JXTA networks. The PeerGroup property in each Action identifies the JXTA peer group where the service is invoked; in terms of integration infrastructure objects, the PeerGroup property identifies the ServiceManager object that should be used to create Service objects.

In the following pages, figures 7.48 through 7.53 illustrate a possible engineering process for buyers, and figures 7.54 through 7.59 illustrate the corresponding engineering process to be run at suppliers. These processes address the five main phases described in section 4.4.3, and they basically resemble the engineering processes shown in figure 7.45, but now in greater detail. One of the main

differences is that internal activities are represented by two consecutive Actions: one that assigns the activity, and another that requests its execution.

The buyer-side engineering process begins with such an activity. As shown in figure 7.48, there is a place named "Assign activity" which is associated with an SMR Action. This Action uses the LRAS service to send a multicast request within the peer group of internal resources. The TargetRole property could specify a role that the desired resource should be able to perform. The MaxHits property was set to 1, meaning that as soon as the SMR Action receives the first reply, it generates an EVT_RESULTS Event; in this case, the Event's SourcePeers property will contain just a single endpoint address. This endpoint address is supplied to the following Action, which will send a request to the corresponding resource.

The Place named "Describe needs" is associated with an SUR Action, which uses the LRTS service to send a task to a given resource. The Action's TargetPeer property is assigned the value of the SourcePeers property, so that the request is sent to the same resource which replied to the previous multicast request. The task involves two documents - TechnologyRequest and BuyerInfo - for which no location is provided since they are supposed to be created by the resource - that is, in fact, the whole purpose of this task. Besides these properties and documents, the SUR Action was given a custom property named LocalPeer1; this property is intended to store the endpoint address of the current resource so that this resource can be invoked again in forthcoming activities, if necessary.

The two Places "Assign activity" and "Describe needs" are associated with Actions that play the role of a LRAS client and a LRTS client, respectively. For a resource to receive these requests and to be able to reply to them, it must have an application that behaves both as an LRAS provider and as an LRTS provider. The workflow client shown in figure 7.47 on the following page is such an application. When the application starts, it requests the user to provide the name of a role that the user is able to perform; then, the application creates two Service objects and starts those Services. When a message is received through LRAS, the application retrieves the value of the TargetRole property and checks if this value matches the role of the user. If so, the application waits a number of seconds, which equals the number of tasks that are still to be performed, and then sends a reply. This ensures that the first workflow client to respond is the one with the lowest number of allocated tasks. When a message is received through LRTS, the application simply adds a new task to the worklist, regardless of whether it has previously replied to an LRAS request or not.

The two Actions that invoke LRAS and LTRS are repeated several times throughout the buyer-side and supplier-side engineering processes; they occur whenever there is an internal activity to be performed. The SUR Action can generate either an EVT_AFFIRMATIVE Event or an EVT_NEGATIVE Event. This is consistent with the functionality provided by the workflow client, which allows the user to create an affirmative or negative reply, as shown in figure 7.47. For the first internal activity of the buyer-side engineering process, which is represented in figure 7.48, only an affirmative reply is expected. If a negative reply is returned, the process does not proceed; in fact, if no purchase needs can be identified, then it makes no sense to proceed with the engineering process further. In other situations, as it can be seen at the bottom of figure 7.49, for example, an affirmative or negative reply may lead to different behaviors.

The remaining Actions deal with services that allow the enterprise to interact with external business partners. To invoke the TPSS service from within the Workflow Kernel, the service client uses the SMR Action to send a request, as shown in figure 7.49, whereas service providers use RMS to receive the request, as shown in figure 7.54. It should be noted that the service client must send a TechnologyRequest document within the request message, according to the definition of TPSS. When a service provider receives a request, it must forward that request to an internal resource, which will

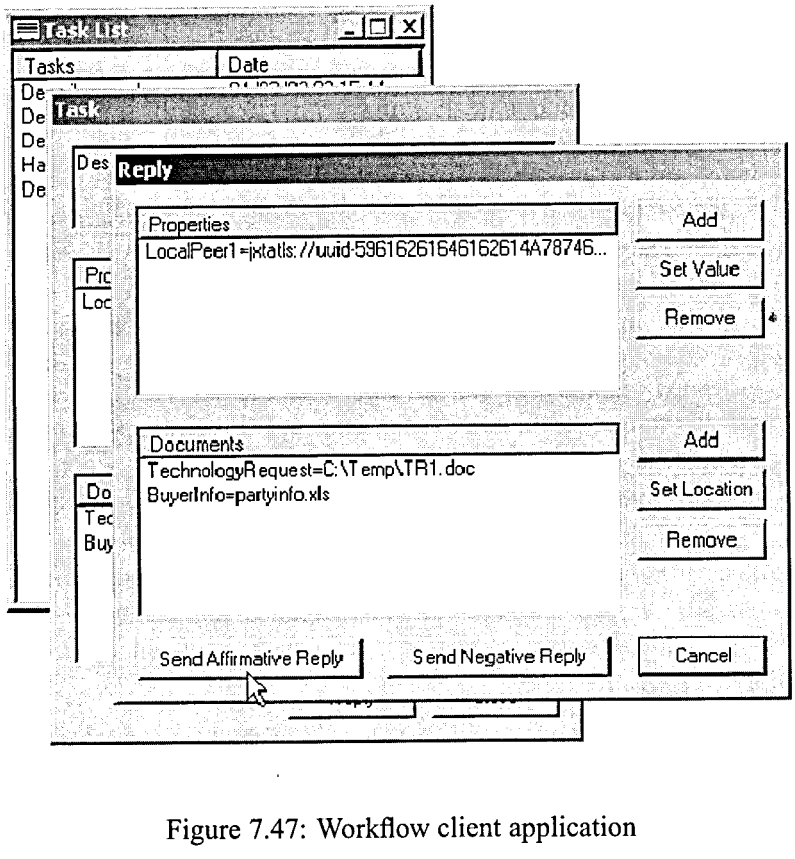


Figure 7.47: Workflow client application

analyze the request and decide whether a reply should be sent or not. This internal activity is shown at the top of figure 7.55. The activity takes the TechnologyRequest document as input, and produces the TechnologyOffer document as output. Afterwards, as shown in figure 7.55, an SSR Action sends the TechnologyOffer document back to the service client.

Going back to figure 7.49, it can be seen that the TPSS client either receives some reply or receives none. In this last case, it may be appropriate to revise the TechnologyRequest in order to match the products available on the market; if this can be done, then partner search will be attempted again. It should be noted that, in order to make the "Revise needs" task be performed by the same user that performed the "Describe needs" task, the TargetPeer property is assigned the value of the LocalPeer1 property, which contains the endpoint address of that user.

If the TPSS client does receive some TechnologyOffers, then a new internal activity, called "Pre-selection", will choose the relevant or interesting offers. As this activity is being assigned, the Suppliers property takes the value of the SourcePeers property which, at this point, contains the endpoint addresses of all TPSS replies. The "Pre-selection" activity takes the TechnologyRequest document and all TechnologyOffer documents as inputs, and it will identify the interesting TechnologyOffer documents, as well as specify the endpoint addresses of the corresponding suppliers. If no appropriate TechnologyOffer can be identified, the user returns a negative reply and the process goes back to the "Revise needs" activity; otherwise, the process proceeds as shown in figure 7.50.

The buyer will now invoke an SMUR Action to request further information from the relevant suppliers, which are indeed waiting for such a request, as shown at the bottom of figure 7.55. The buyer sends the BuyerInfo document, which has been produced earlier as an output of the "Describe

needs" activity, and expects each of those suppliers to return a SupplierInfo document, according to the definition of TPIS. When a supplier receives the TPIS request, it starts an internal activity to analyze the BuyerInfo document and to produce a SupplierInfo document, as shown in figure 7.56. As soon as this activity is completed, the supplier sends the SupplierInfo document back to the buyer by invoking an SSR Action. If the supplier finds that in spite of having returned a TechnologyOffer it cannot actually cope with the buyer's requirements, the "Analyze and produce info" activity returns a negative reply, and the supplier produces no TPIS reply.

The SMUR Action shown at the top of figure 7.50 will eventually generate an Event, either because all suppliers have replied or because a certain number of seconds has elapsed. In both cases, the buyer forwards the received SupplierInfo documents to an internal activity called "Selection", whose purpose is to choose the ultimate supplier. If there is no satisfying supplier, then the process goes back to the "Revise needs" activity shown in figure 7.49. Otherwise, the "Selection" activity returns the chosen TechnologyOffer, the corresponding SupplierInfo document, and the endpoint address of the supplier; then the process proceeds as shown in figure 7.51. Basically, the buyer prepares a TPA document and sends it as a proposal to the chosen supplier. The buyer sends the TPA as ContractProposal document and expects the supplier to return a ContractResponse document, according to the definition of TPAS.

At the supplier side the process is waiting for a TPAS request, as shown at the top of figure 7.57. Once it is received, the ContractProposal document is forwarded to an internal activity which will analyze the TPA proposal and will produce a response. If the TPA proposal is accepted, then the ContractResponse document is the same as the ContractProposal document, and the "Analyze contract" activity returns an affirmative response. If the TPA proposal is not accepted, then the "Analyze contract" returns a negative response together with a ContractResponse document containing a TPA proposal that the supplier would prefer. In both cases, the ContractResponse document is sent to the buyer by means of an SSR Action. However, if a negative reply has been issued, the supplier waits for another TPAS request from the buyer, as shown in figure 7.57. This loop can repeat itself until the buyer submits an acceptable proposal or until the buyer gives up trying to reach a TPA with this supplier.

At the buyer side the process waits for the TPAS reply from the supplier, as shown at the top of figure 7.52 (which is the same as what is shown at the bottom of figure 7.51). If the reply is negative, the buyer initiates an internal activity called "Revise contract", which is assigned to the same resource that performed the "Prepare contract" activity. The ContractProposal document is both an input and output document for the "Revise contract" activity, whose purpose is to modify the original TPA proposal according to the supplier's ContractResponse document. When this activity is completed, a new TPAS request is sent to the chosen supplier. This loop repeats itself until the supplier accepts a TPA proposal. Alternatively, the buyer may give up trying to reach a TPA with this supplier and it may prefer to choose another supplier. This situation has not been modeled in figure 7.52 but what it means is that, should the "Revise contract" activity return an EVT_NEGATIVE Event, then the process should go back to the selection phase.

When the supplier accepts the TPA proposal, both processes run into an internal activity called "Configuration", as shown in figures 7.52 and 7.58. At each end, the purpose of this activity is to define and configure the operational process that will be run during the fulfillment phase. Operational processes are governed by two kinds of requirements: on one hand, they are subject to the way each enterprise performs its own internal processes; on the other hand, they are subject to the way both buyer and supplier agreed to perform them. It should be noted that as long as both buyer and supplier comply with the message exchange behavior they have agreed to, operational processes may

comprise internal activities that have not been described in the TPA. Formally, a private workflow can be implemented in a different way from the corresponding public workflow as long as the WF-net that describes the private workflow is a *subclass* of the WF-net that describes the public workflow - this concept is known as *workflow inheritance* [Aalst, 2000a].

For both the buyer and supplier, the operational process runs as a sub-process within the engineering process. The buyer-side operational process is called "Buyer Process", as shown at the top of figure 7.53; the supplier-side operational process is called "Supplier Process", as shown at the bottom of figure 7.58. Both of these are names of processes that have been configured within their respective Workflow Kernel. Whenever a Token reaches a Place that is associated with a sub-process, the specified sub-process is started, i.e., a new Token is inserted into that Process's input Place. The sub-process is started for the same color as the Token in the parent process. As soon as the operational process finishes, it generates an NTF_PROCESSCOMPLETE Event and, within the engineering process, the Transition associated with this Event fires, taking the engineering process into its last and final internal activity, as shown in figures 7.53 and 7.59, respectively.

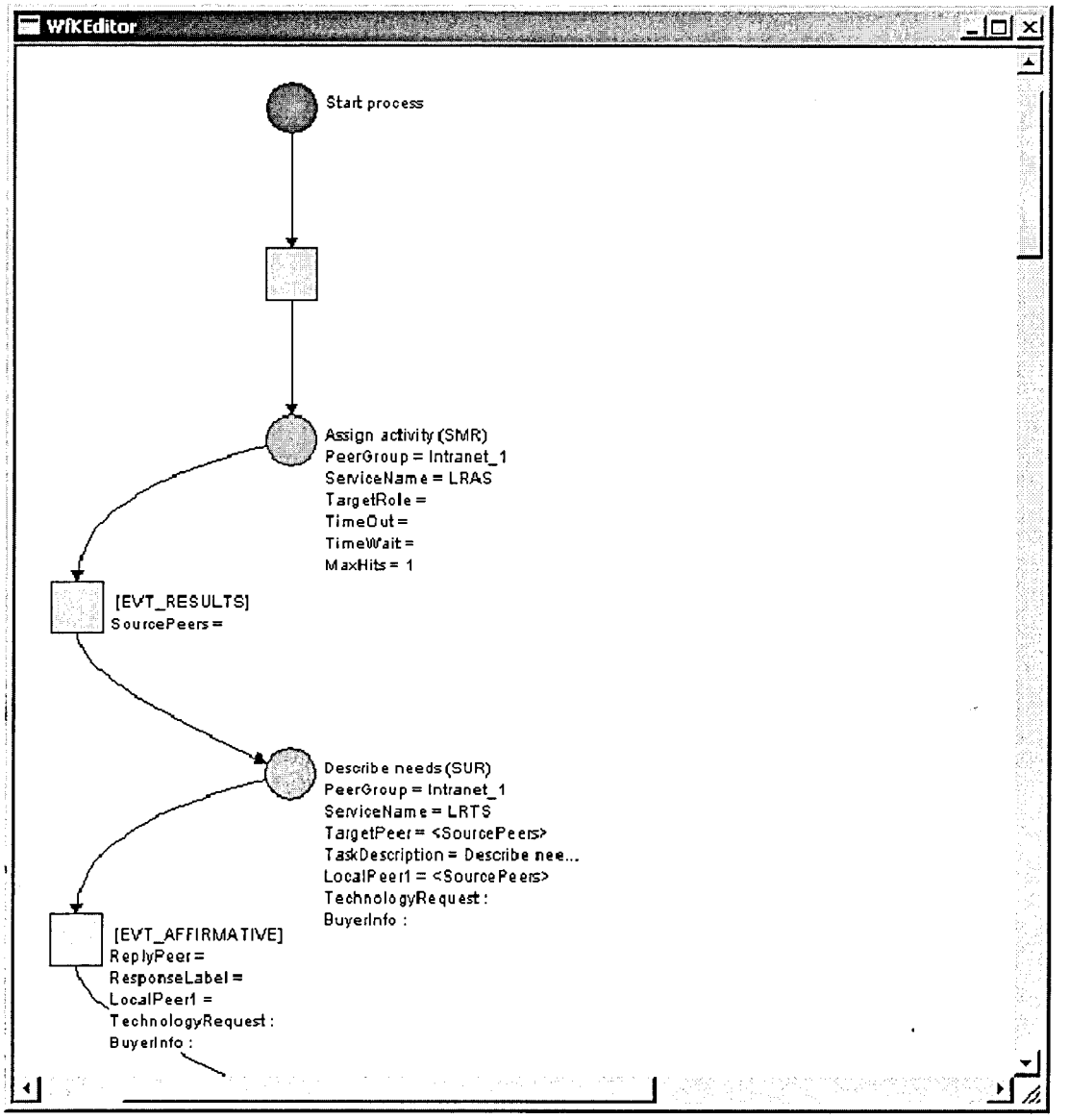


Figure 7.48: The buyer-side engineering process

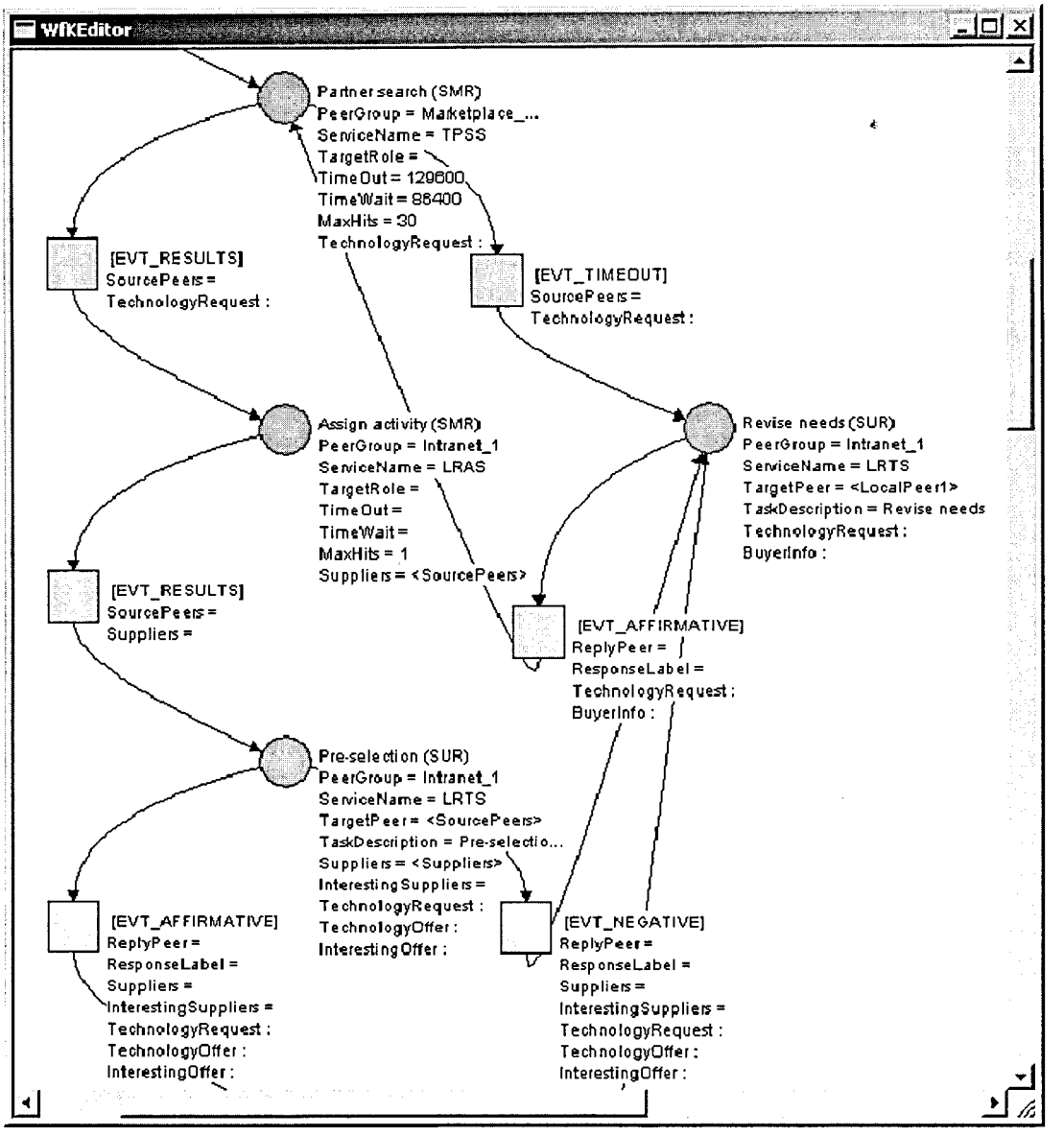


Figure 7.49: The buyer-side engineering process (continued)

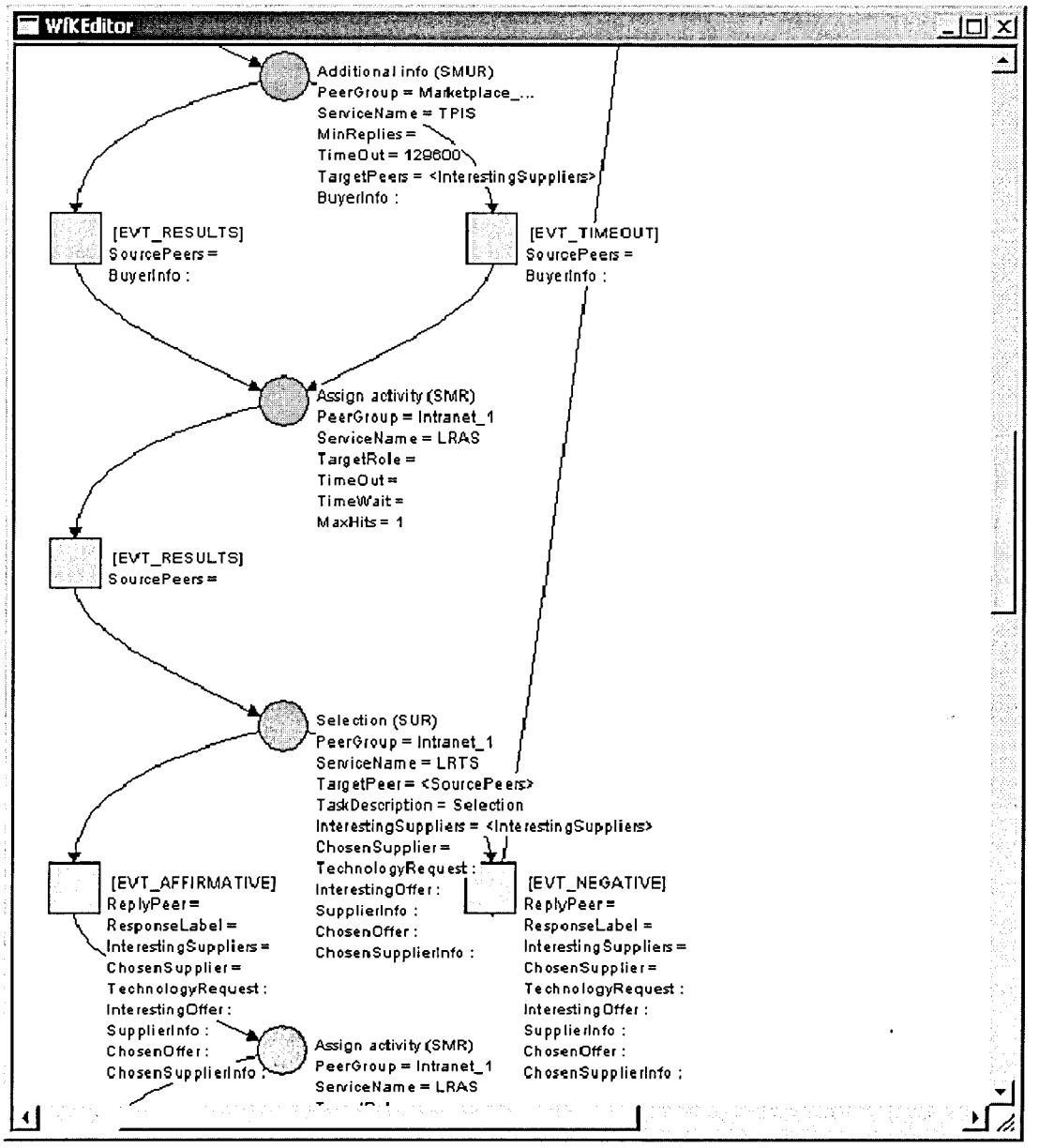


Figure 7.50: The buyer-side engineering process (continued)

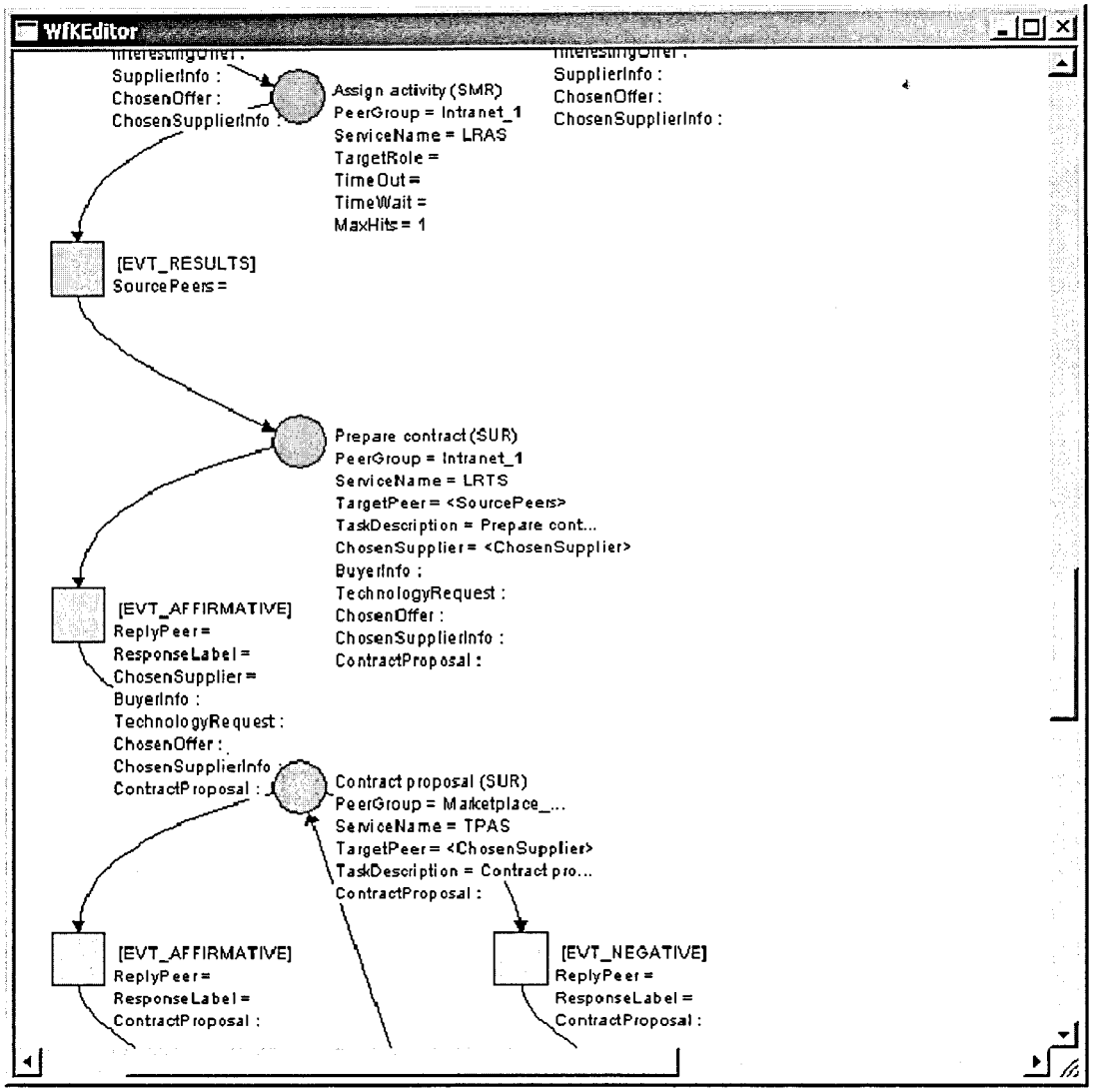


Figure 7.51: The buyer-side engineering process (continued)

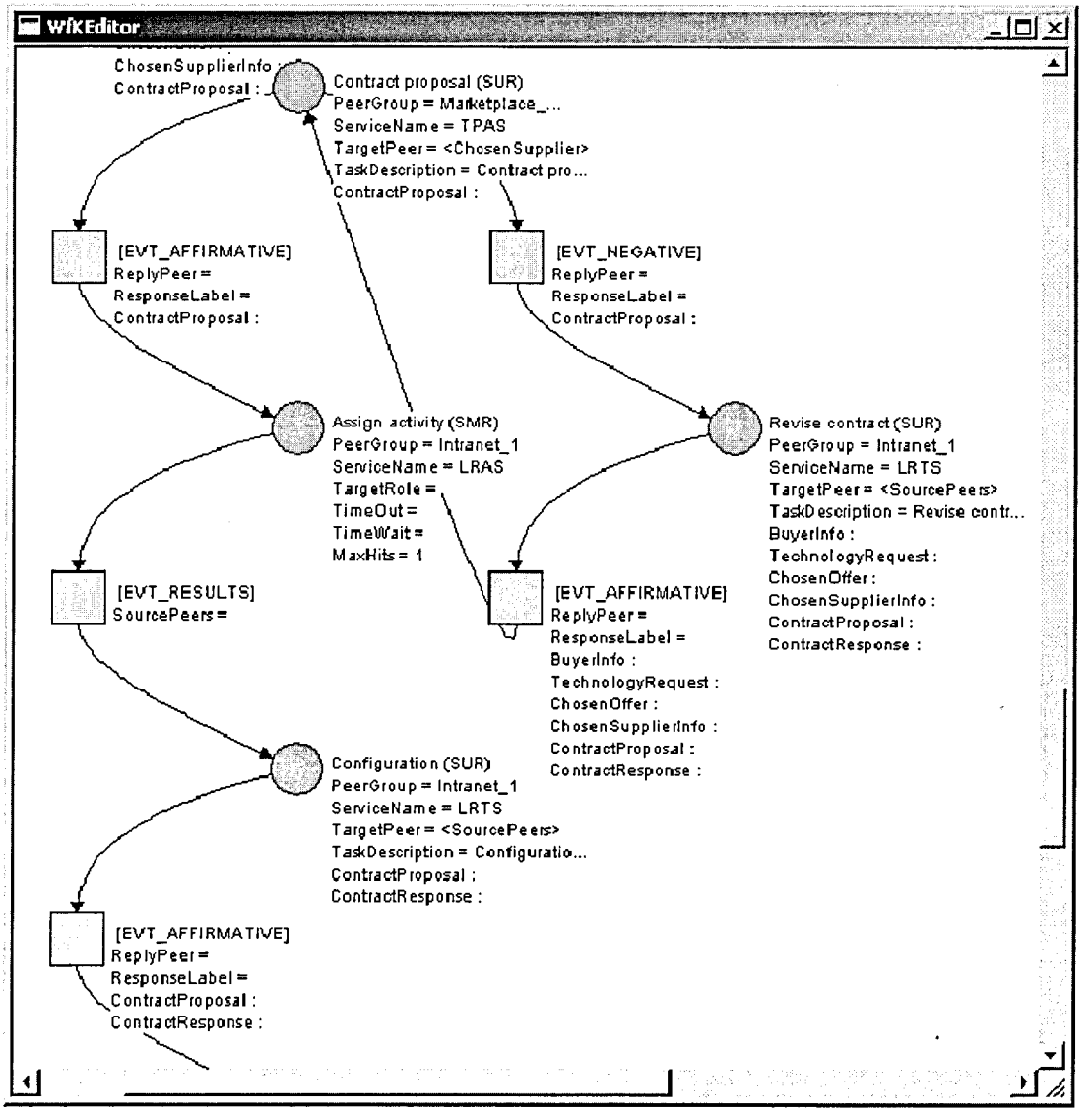


Figure 7.52: The buyer-side engineering process (continued)

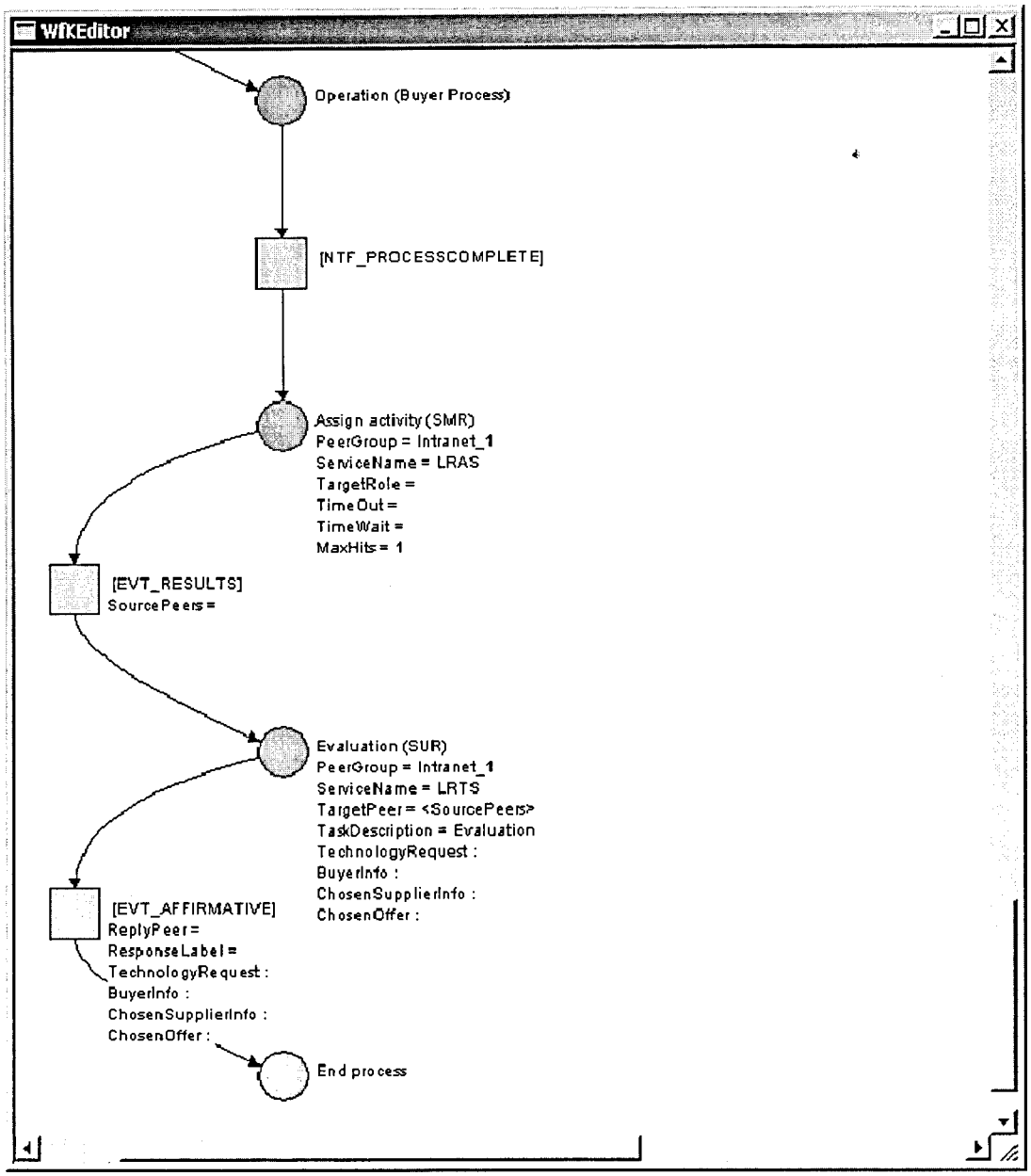


Figure 7.53: The buyer-side engineering process (continued)

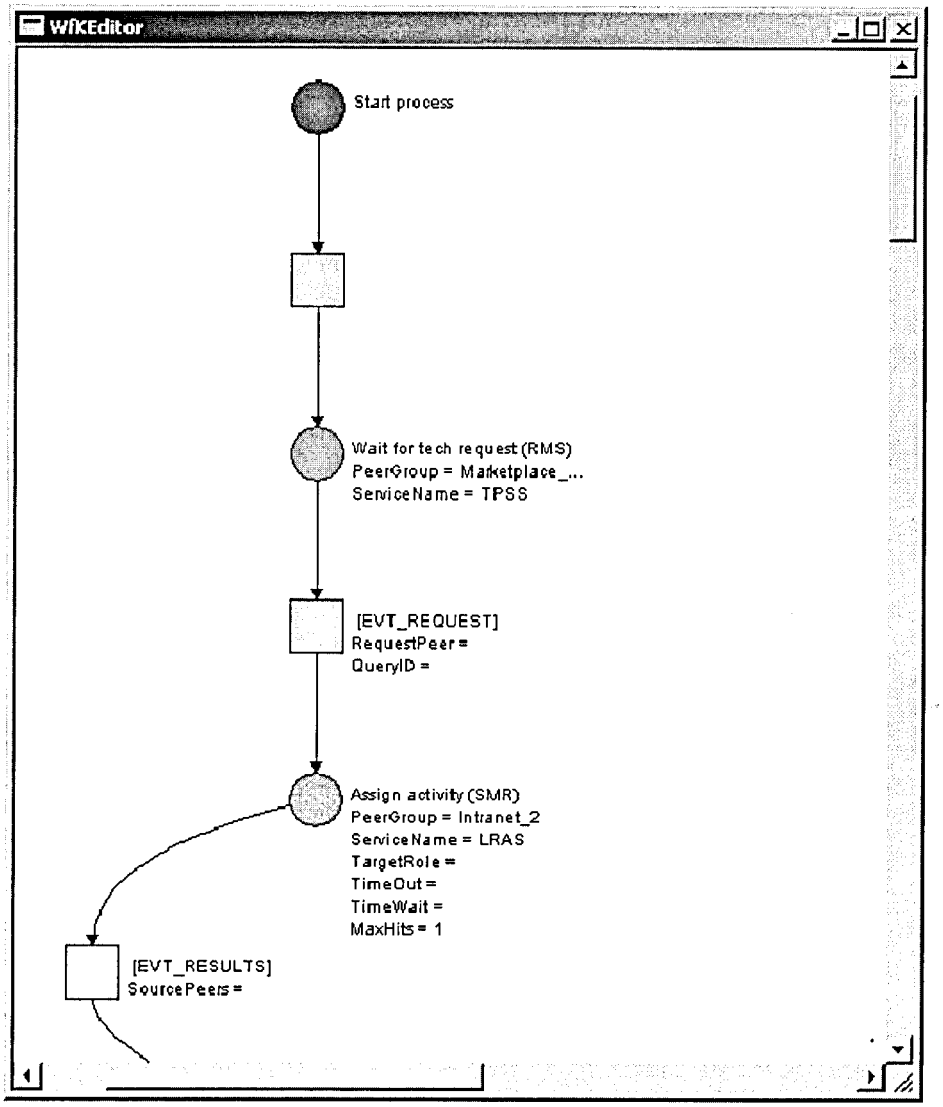


Figure 7.54: The supplier-side engineering process

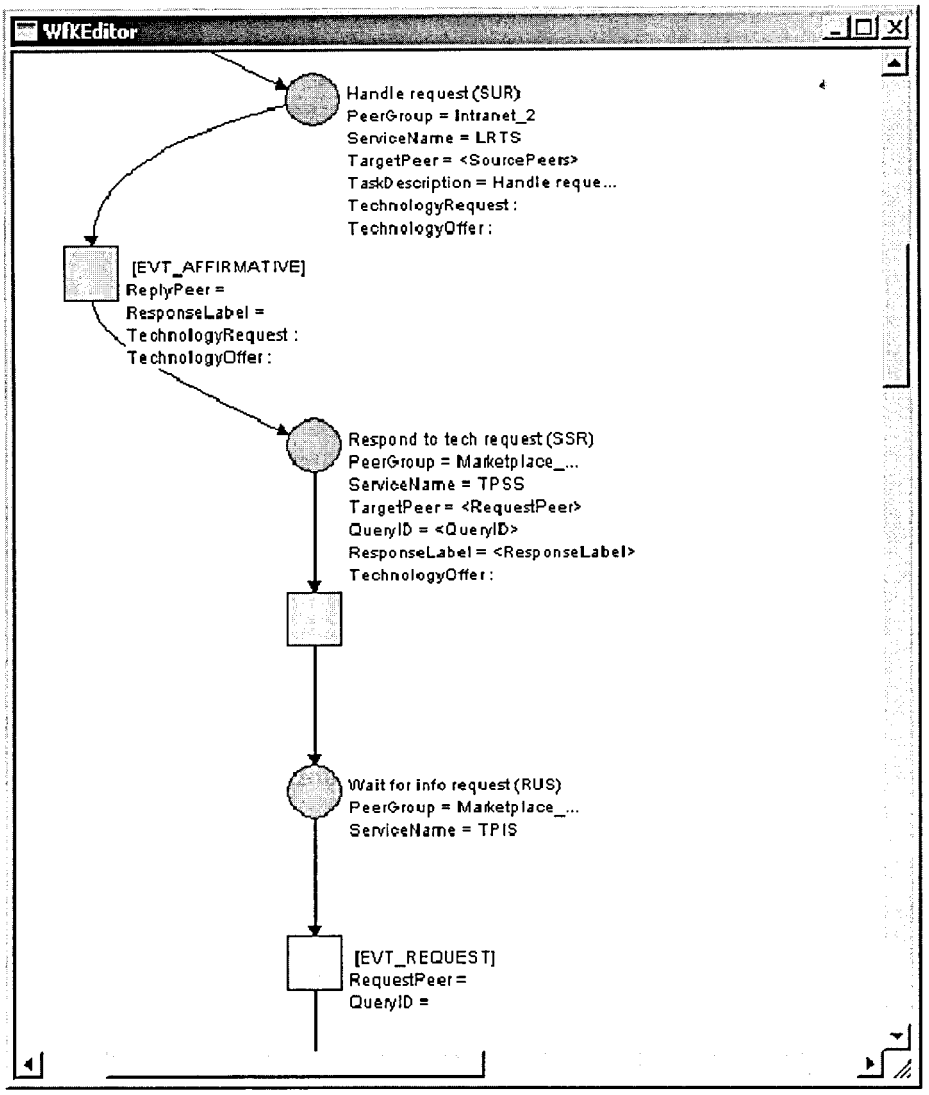


Figure 7.55: The supplier-side engineering process (continued)

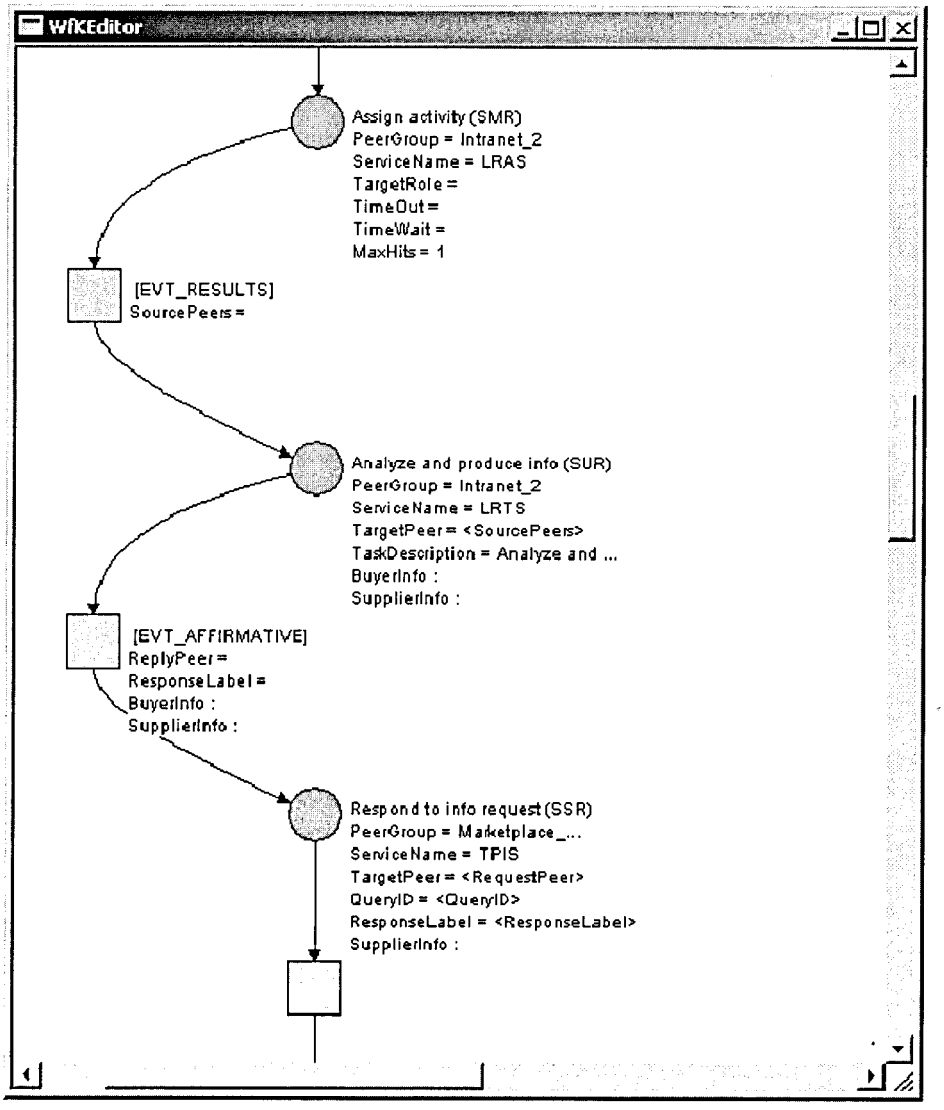


Figure 7.56: The supplier-side engineering process (continued)

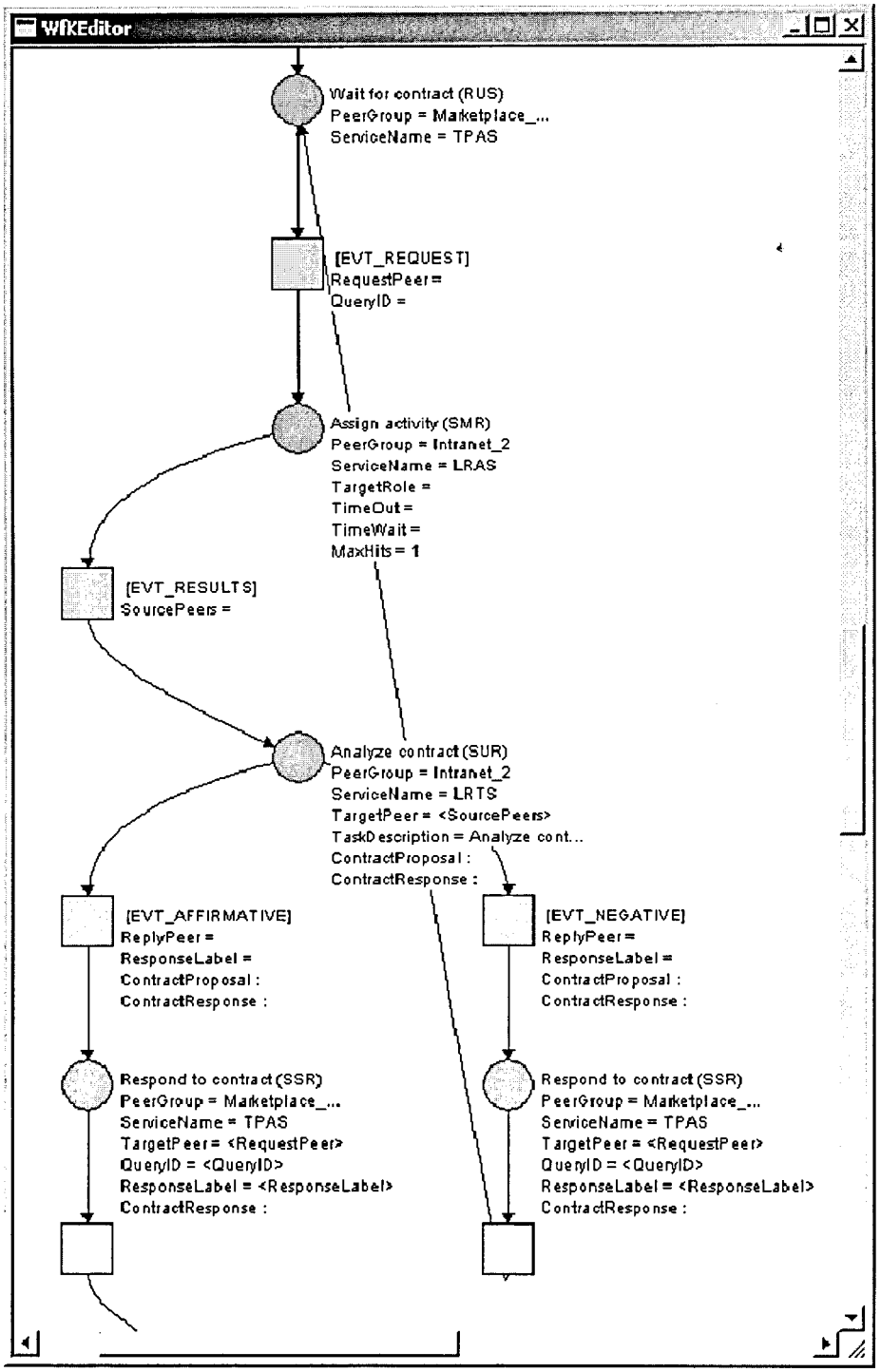


Figure 7.57: The supplier-side engineering process (continued)

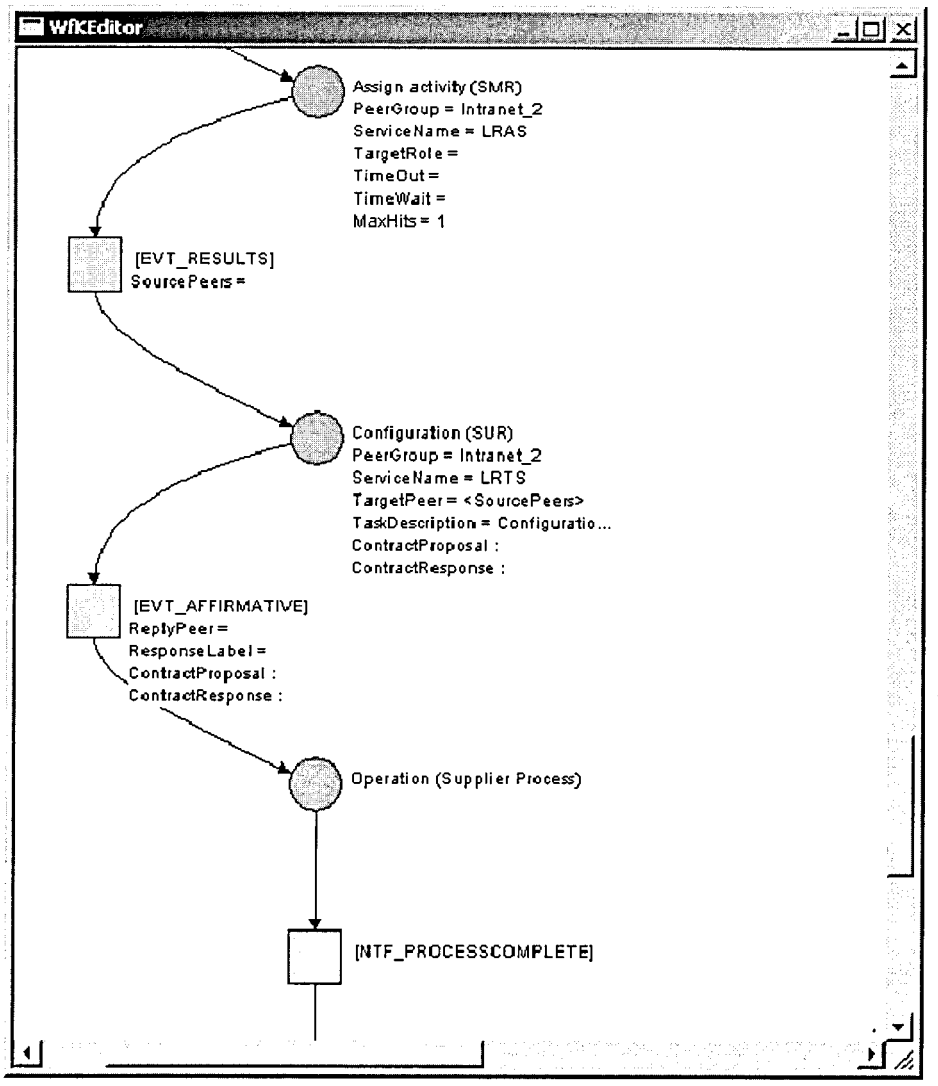


Figure 7.58: The supplier-side engineering process (continued)

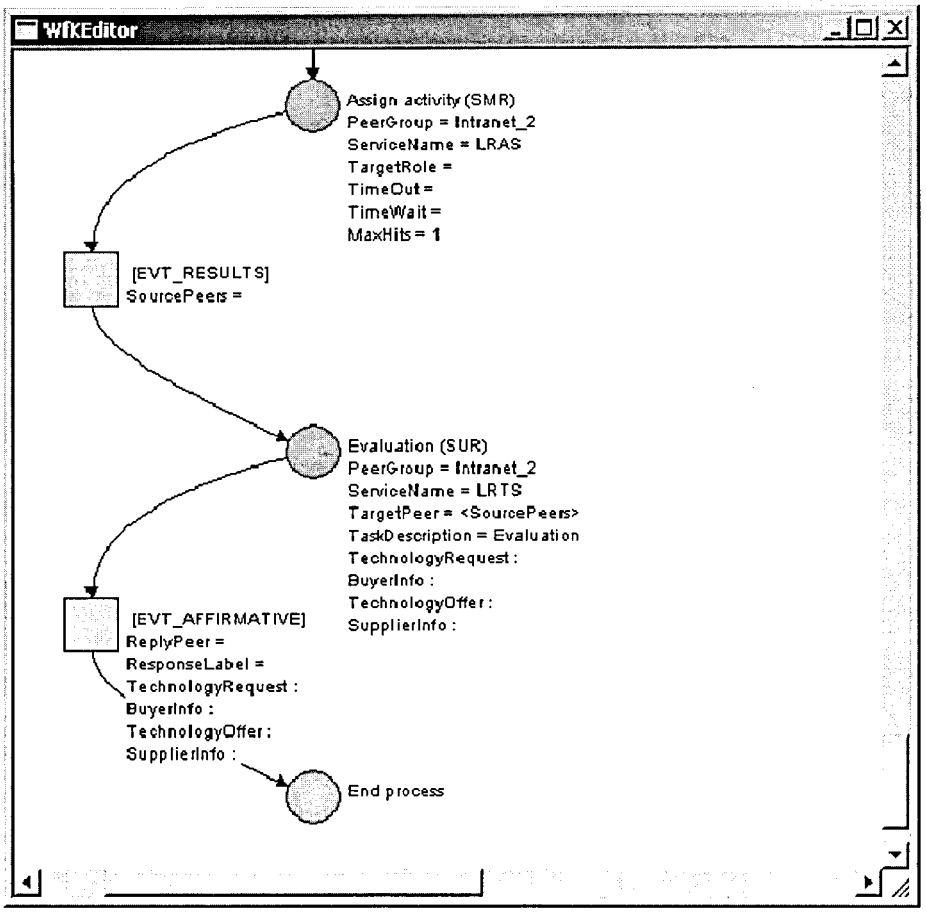


Figure 7.59: The supplier-side engineering process (continued)

7.3.5 Business networking

The previous section has shown how workflow-controlled actions and P2P services can be combined to support the life cycle of a business relationship between two enterprises. This approach can be generalized to the case of an arbitrary number of enterprises establishing relationships with each other. A major advantage of this fruitful combination is that workflow management provides enterprises with the flexibility to define their behavior, whereas P2P networking ensures that enterprises have the autonomy and freedom to build relationships with any available business partner. The result is a marketplace where enterprises can link their processes together and develop arbitrary business structures.

Figure 7.60 illustrates how this approach can support a business structure such as a virtual organization. Basically, a virtual organization is a conglomerate of independent partners that cooperate in order to offer an integrated product that none of the partners would be able to produce individually. The engineering process for the virtual organization must address the need to find business partners with certain competencies. Before the operational process can be settled, these business partners have to be found, selected, and contracted. For this reason, the engineering process breaks up into several parallel paths, each having its own search phase, selection phase and contracting phase. The operational process for the virtual organization must be configured according to the TPAs that have been established with each partner.

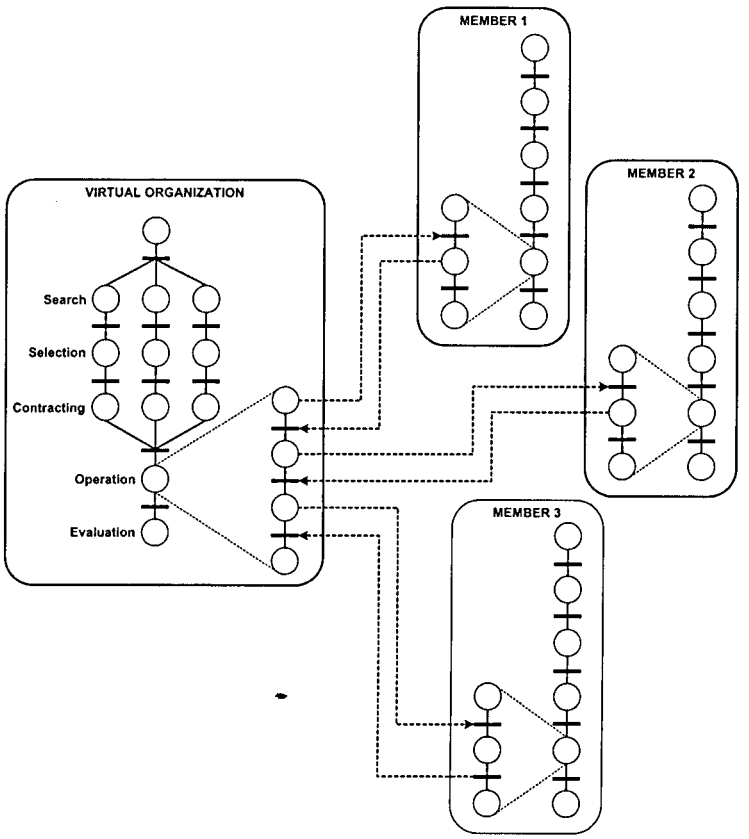


Figure 7.60: Business relationships within a virtual organization

Figure 7.60 also highlights the fact that the operation of virtual organizations relies on a central, coordinating entity. Nevertheless, it should be possible for enterprises to establish relationships with each other without being subject to the role of a third party. In addition, enterprises should be free to develop and maintain their relationships without being restricted to particular consortia. If each enterprise has its own set of relationships with external business partners then, when two enterprises establish a new relationship between them, they are also establishing an indirect relationship between their business partners. The set of all enterprises connected by direct or indirect relationships constitutes a business network. According to this line of reasoning, it can be argued that a virtual organization is just a special case of business network.

A more generic form of a business network is illustrated with an example in figure 7.61. Here, enterprise 1 establishes a relationship with enterprise 2 which, in order to perform the operational process, must rely on services provided by enterprise 3. On its turn, enterprise 3 has a business relationship with enterprise 4. For each of these relationships there is one peer that plays the role of buyer and another peer that plays the role of supplier. Enterprise 2 is a supplier for enterprise 1 and a buyer for enterprise 3; in a similar way, enterprise 3 is a supplier for enterprise 2 and a buyer for enterprise 4. This means that the engineering process running at each peer may have to simultaneously support different roles. This can be done simply by interleaving service-provider actions (to interact with buyers) with service-client actions (to interact with suppliers) within the same process.

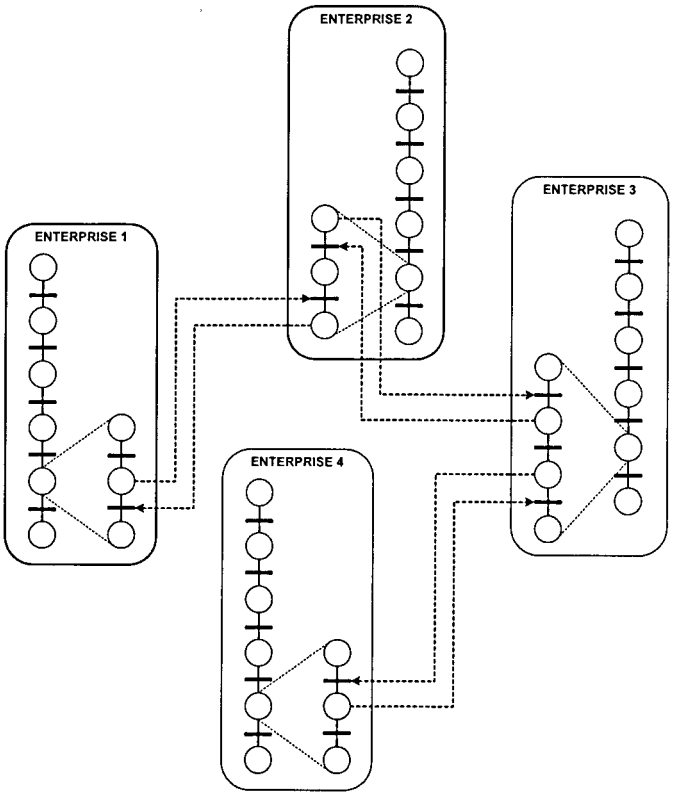


Figure 7.61: Business relationships within a business network

7.4 Concluding remarks

The proposed solution described in this chapter defines the basic building blocks - Actions and Services - that allow enterprises to integrate their business processes with each other. The proposed solution also provides the run-time system - the Workflow Kernel and the P2P integration infrastructure - that supports the execution of those processes. The result is a workflow management system that departs from the workflow solutions discussed in previous chapters. At the integration infrastructure level, workflow participants - both external business partners and internal resources - are not required to connect to a central server, nor to rely on services provided by third parties. At the workflow enactment level, there is no single process that describes the whole set of exchanges between a group of business partners; rather, workflow-automated behavior is achieved by synchronizing processes running at different enterprises.

Even so, the proposed solution described in this chapter requires minimum coupling between the workflow enactment services running at different enterprises. For workflow enactment services to be able to exchange messages with one another, all they have to know is the name of a multicast service or, alternatively, they must know the name of a unicast service plus a destination endpoint address. It should be noted that, in general, endpoint addresses are found by first issuing a request via a multicast service. Therefore, the ultimate link between two workflow enactment services is simply a set of service names. The advantage is obvious: connecting enterprises together can be done with minimal effort. All that business partners have to define is the use of certain services, the documents associated with the services, and their own behavior, which can be configured by means of appropriate Actions inserted in Processes running at both ends.

This approach is in sharp contrast with other workflow management systems, which often require workflow enactment services to implement a standard API in order to become interoperable with each other; this is illustrated by interface 4 of the Workflow Reference Model (section 3.3.1.4 on page 74). Interface 4 defines a set of interfaces that workflow enactment services should implement; these interfaces allow a workflow enactment service to create, start, terminate and retrieve information about process instances running at a remote workflow enactment service. Clearly, this approach is not suitable for an inter-enterprise scenario because no enterprise can be given permission to control the execution of business processes running at another, independent enterprise. This issue is further exacerbated by the fact that both enterprises may have been previously unknown to each other. Whenever an enterprise receives a message from another enterprise, it is the first enterprise and that enterprise only who can decide to start a new process instance or determine how this message affects running process instances. Therefore, a relationship between enterprises is more likely to be expressed in terms of a message exchange sequence, rather than in terms of the workflow activities that are carried out at each enterprise.

This result is somewhat counter-intuitive. The work described throughout this thesis seems to be the quest for process integration between enterprises but, after all, as one approaches the proposed solution, one finds that it is not appropriate to create explicit links between processes running at different enterprises. To understand that this in fact no paradox, one must realize that two processes can indeed be connected to each other: for example, placing an SMR Action in one process and an RMS Action in another process and configuring both Actions with the same peer group and the same service name effectively means that both processes are linked. What is lacking is a view which would allow an enterprise to look at the processes running at its business partners. If one considers that each enterprise must have the freedom to implement its own business processes in any desired way and to establish business relationships with any available business partner and according to any business

structure, then one will eventually realize that such a centralized view goes against the autonomy that each and every enterprise must be granted.

Like the proposed solution described in this chapter, a workflow management system that supports the engineering of business networks must be a decentralized system both at the integration infrastructure level and at the workflow execution level. At the integration infrastructure level, a peer-to-peer networking platform is the ideal solution because: it allows enterprises to discover and interact with each other, it cannot be brought under the control of a single entity, and its value and potential increases dramatically as more enterprises connect to the same infrastructure. At the workflow execution level, each enterprise should have its own workflow enactment service in order to define its own processes and configure automated exchanges with external business partners. The Workflow Kernel is a reusable workflow enactment service, which is intended to provide essential workflow capabilities for any workflow management system in general.

The resulting integration architecture is illustrated in figure 7.62. Provided with a workflow enactment service, each enterprise can define and execute its own business processes, as well as the interaction between these processes and the business processes running at other enterprises, in order to automate a complete business-to-business trading process such as the one described in section 4.4.3 and presented in more detail in section 7.3.4. Provided with a service-based integration infrastructure, together with a set of specific Actions, enterprises can interact using decentralized, peer-to-peer networking mechanisms. The proposed workflow solution is said to support the engineering of business networks because it allows each enterprise to define and to automate the whole business-to-business trading life cycle, thereby supporting part of engineering life cycle of the enterprise itself.

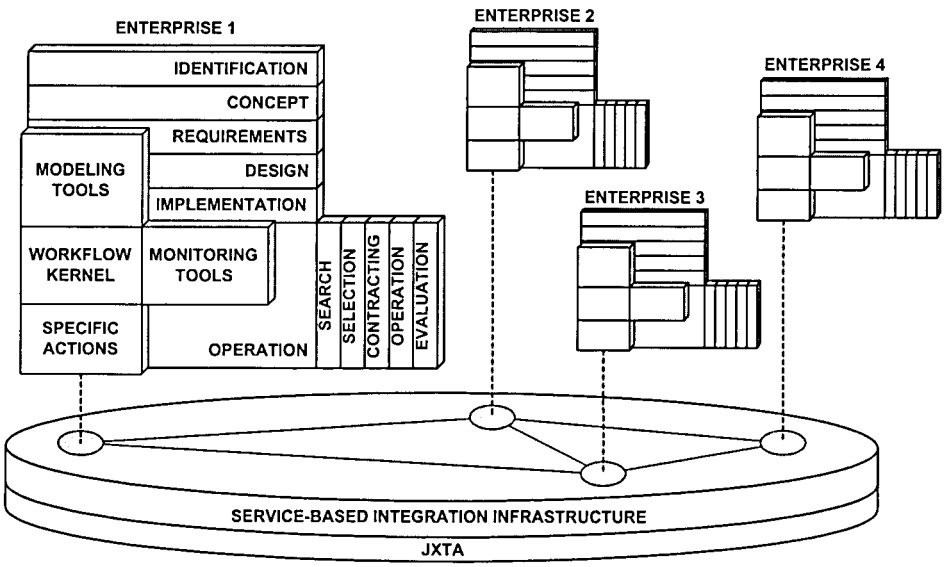


Figure 7.62: Proposed integration architecture for business networks

Chapter 8

Conclusion

From an initial assessment of the impact of the Internet on business practices, to the presentation of several systems, projects and approaches that aim at integrating business processes within and across enterprise borders, this work shows that Information Technology opens new possibilities for companies to restructure both their processes and their organization. In particular, workflow management systems, which have been typically employed in intra-enterprise environments, can play a very important role in inter-enterprise environments by allowing an enterprise to coordinate its own business processes with those of its business partners. In an inter-enterprise environment, where business is structured according to a network of competencies provided by a set of autonomous entities, workflow management systems become a valuable tool to automate the exchanges that allow these entities to discover, develop and implement business relationships with each other. The solution proposed in this thesis illustrates how this life cycle can expressed as a process model and then released for execution using an appropriate workflow management system.

8.1 Summary of main contributions

Overall, the work described in this thesis has led to a number of contributions. Some of these contributions concern workflow management systems in general, while others concern the application of workflow management systems to the engineering of business networks. In particular, this work:

- *Shows that a workflow management system must be devised as a combination of a workflow enactment service and an integration infrastructure* - Typically, workflow management systems address process modeling and execution, and pay less attention to the need for integrating all resources involved in the business processes they deal with. But having an appropriate integration infrastructure is essential in order to take full advantage of workflow modeling and execution capabilities. The workflow enactment service is able to model and control the execution of arbitrary process behavior; the integration infrastructure makes sure that any specified behavior, which requires the exchange of information between a set of resources, can indeed take place.

- *Shows that the observer pattern is at the core of every workflow activity* - When a workflow activity begins, the workflow enactment service dispatches the input to the assigned resource and then it waits for that resource to send back one or more outputs. During activity execution, the workflow enactment service becomes an observer of the outputs produced by the assigned

resource. This means that the underlying integration infrastructure must also support the observer pattern. In general, the observer pattern may have to be implemented across several software layers and even across the network, depending on the ultimate location of resources. In practice, the observer pattern is often replaced by the use of a common parameter value, which relates a reply to a previous request.

- *Shows that the development a workflow management system requires the integration of different technologies* - Given that a workflow management system is a compound of two separate components, combining these components may require the integration of different technologies. The purpose of the workflow enactment service is to control the execution of workflow processes, while the purpose of the integration infrastructure is to promote interoperability between the available applications and resources. Each of these components may be built using a different technology: the workflow enactment service requires a technology that allows the user to model processes and monitor process execution, whereas the integration infrastructure requires a technology that is able to integrate heterogeneous applications.

- *Provides a reusable workflow enactment service based on Petri nets* - Rather than developing a third workflow enactment service, this work has tried to come up with a generic set of workflow functionality supporting process modeling and execution. To ensure that such a workflow enactment service would be independent of any particular workflow system architecture, the system makes use of Petri nets and extends them with the concepts of Actions and Events. Petri nets provide a solid foundation, formal semantics and pave the way for process analysis and verification; the concepts of Actions and Events insulate the system from particular mechanisms of resource invocation. By means of a set of external interfaces, the resulting workflow enactment service can be reused or embedded in other workflow management systems.

- *Provides a completely decentralized, service-based integration infrastructure* - In the quest for an integration infrastructure to be used in inter-enterprise environments, this work was led to the conclusion that only a decentralized platform like a peer-to-peer system can enable different enterprises to interact without compromising their autonomy. Built on JXTA, the proposed integration infrastructure is a generic messaging platform that allows peers to organize themselves into peer groups and to communicate with each other using unicast and multicast services. Peers are able to define common services that they will use for some specific purpose. To support the engineering of business networks, four services have been defined, which allow enterprises to discover each other and to establish business relationships with each other.

- *Provides a solution that requires minimum coupling between enterprises* - Using a common set of services, enterprises can discover and exchange messages with each other without having to expose any programmatic interfaces. In general, a service has a unique name and a set of associated documents to be included in service requests and replies. The service client is the party who sends a request and receives replies; the service provider is the party who receives a request and produces a reply. Within a business relationship, each enterprise may play the role of both service client and service provider at different times. The behavior of a service client and that of a service provider can be easily defined within a workflow process by inserting appropriate Actions.

- *Provides a solution that allows enterprises to keep their internal processes private* - Each enterprise is free to handle service requests and replies in any desired way. What a service provider

does between the moment it receives a request and the moment it issues a reply, or what the service client does before sending a request and after receiving a reply, depends on internal processes only and does not concern the other party. When two enterprises establish a business relationship they agree on a common, network-level process, which describes how they will interact with each other. From this network-level process, each party can configure its workflow processes in order to blend the required network-level behavior with its own internal activities.

- *Provides a solution that can support both intra- and inter-enterprise processes* - The kind of decentralized integration infrastructure that has been devised for inter-enterprise environments can be effective within a single enterprise as well. In this case, each workflow participant becomes a peer in a private P2P network. Provided with a simple workflow client application, which responds to a pre-defined set of services, workflow participants can receive tasks and return results. To support both role assignment and user assignment, two services have been defined. Only the workflow processes running in the workflow enactment service establish the link between the inter-enterprise, public environment and the intra-enterprise, private environment.

- *Provides a solution that automates the business-to-business trading life cycle* - Whereas workflow management systems have been typically employed to coordinate a set of routine tasks, this work has shown that it is possible to apply workflow management to the whole trading life cycle, from the partner search phase to the evaluation phase. In an increasingly dynamic marketplace, the automation of the business-to-business trading life cycle ensures that business relationships will develop according to a definite sequence of steps; it also enables an enterprise to quickly retrieve the status of all business relationships with other enterprises, or to monitor the development of those relationships; but first and foremost, it improves the ability of an enterprise to establish and manage an increasing number of business relationships with external business partners.

- *Provides a solution that partly supports the enterprise engineering life cycle* - The proposed workflow management system includes process modeling capabilities that allow enterprises to describe how they will discover and interact with each other. Using sub-processes and the ability to nest sub-processes within processes, enterprises can also build partial models for certain types of behavior. Should enterprises change or improve their processes, in consequence of continuous improvement or in consequence of an enterprise reengineering project, they can update or redesign these models and release them for execution using the same process execution capabilities and integration infrastructure. Therefore, the proposed solution outlives particular modes of operation and is useful across the design, implementation and operation phases of the enterprise engineering life cycle.

- *Provides a solution that allows enterprises to develop arbitrary business structures* - The proposed workflow management system allows enterprises to develop pairwise business relationships, which can be regarded as the building blocks of a business network. If all business relationships have a common end then the business network reflects a centralized structure such as that of a virtual organization. But the decentralized nature of the proposed integration infrastructure, together with the fact that every enterprise maintains control of its own processes by means of its workflow enactment service, means that it is possible to establish other kinds of business structures. The point is that enterprises can create a business network as an overlay to

the underlying peer-to-peer network. Whether that business network exhibits a centralized or decentralized structure, it is left for enterprises to decide; the purpose of the proposed workflow management system is to support business networking in general.

8.2 Open issues and future work

This work is intended as a research contribution to the field of workflow management systems. In particular, its ultimate goal is to show how workflow management systems can support the engineering of business networks. In the pursuit of this goal, many issues have been discussed, and many more remain to be addressed. In some respects, the work described in this thesis could be improved; on the other hand, it draws the attention to some issues that deserve further research. In future work, the following issues should be considered:

- *Compatibility of engineering processes* - An engineering process unfolds itself as a sequence of interactions using different services. If a new enterprise connects to the integration infrastructure it must know how to configure its engineering processes in order to exhibit a behavior that is compatible with the remaining enterprises. The question is who defines the network-level engineering process, and how this process is made known to other enterprises. The engineering process could be defined by a group of enterprises, who would then publish it as an advertisement.

- *Process analysis and simulation* - The proposed workflow enactment service is based on Petri nets but the possibility of using the described analysis techniques has not been explored further. Given that there are several analysis tools available, the workflow enactment service should be able to export process definitions in a format that can be imported into some of those analysis tools. Also, the simulation capabilities of the workflow enactment service could be improved.

- *Assisted process configuration* - Once two enterprises reach an agreement they must configure their operational processes according to the network-level process defined in that agreement. Rather than requiring a human user to configure the operational process by hand, the workflow enactment service should be able to automatically generate an outline of how the process should look like regarding message exchanges with the other party.

- *Implementation of B2B frameworks* - The proposed solution provides process modeling and execution capabilities which allow the interaction model between a set of business partners to be designed and automated. In addition, the integration infrastructure is a decentralized messaging platform that supports the exchange of XML documents between a set of peers. This suggests that it should be possible to implement some B2B frameworks using the proposed system.

- *Coordination and collaboration services* - The services discussed so far were focused on the exchange of messages that delimit the execution of workflow activities. Other kinds of services, such as, for example, a service that allows the workflow enactment service to retrieve the status of a previously requested task, could be useful both in inter-enterprise and in intra-enterprise environments. Another interesting idea is that, since resources are connected across a P2P network, they could make use of collaboration services such as instant messaging and file transfer.

- *Fault-tolerance and exception handling* - The proposed solution does not provide mechanisms to deal with system failures; these failures could occur at different software layers. In general, however, the most likely sources of problems will be the impossibility of reaching a remote peer or the inadequacy of some requests or replies. To deal with the first problem, not much can be done other than to keep trying; to deal with the second problem, the available Actions should be able to generate an additional type of Event that denotes a message error.

- *Development of trust models* - In a P2P inter-enterprise environment, managing trust becomes an important issue since enterprises will often encounter previously unknown entities. Moreover, in a P2P environment there is no central repository of information that would allow an enterprise to determine whether it should trust a potential business partner or not. Therefore, the proposed solution requires the development of trust models that support trust assessment based on information collected from several sources or dispersed throughout the network.

- *Additional modeling views* - Perspectives from enterprise integration architectures show that workflow management systems can be effectively employed as enterprise integration tools. However, as an enterprise modeling tool, their modeling scope is narrowly focused on business processes. As workflow management systems become capable of supporting business networking and new forms of business organization, they should also become capable of dealing with process definitions that contain elements from additional modeling views, such as data view and organization view.

8.3 Closing statement

This thesis ends at a point where it seemed to have just begun. Now that it has been shown how workflow management systems can support the engineering of business networks, only the future will tell whether enterprises will be interested in such kind of solution or not. The proposed solution fosters a new kind of marketplace - a peer-to-peer marketplace - where enterprises can establish one-to-one relationships with each other, and provides them with the ability to manage and automate their relationships. But such kind of marketplace will only become a reality when it reaches a certain critical mass, which requires many enterprises to connect to a common, global network infrastructure. That infrastructure already exists; now it is just a matter of deploying appropriate solutions on top of that platform. In this context, the work described in this thesis will prove to be, hopefully, an interesting combination of many concepts and technologies.

References

AALST, W. VAN DER, "The Application of Petri Nets to Workflow Management", Journal of Circuits Systems and Computers, vol. 8, no. 1, pp. 21-66, 1998

AALST, W. VAN DER, "Inheritance of Interorganizational Workflows: How to agree to disagree without loosing control?", Technical Report CU-CS-899-00, University of Colorado, Department of Computer Science, Boulder, USA, 2000

AALST, W. VAN DER, "Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques", in "Business Process Management: Models, Techniques, and Empirical Studies", Lecture Notes in Computer Science, vol. 1806, pp. 161-183, Springer-Verlag, 2000

AALST, W. VAN DER, HOFSTEDE, A. TER, KIEPUSZEWSKI, B., BARROS, A., "Workflow Patterns", BETA Working Paper Series, WP 47, Eindhoven University of Technology, Eindhoven, Netherlands, 2000

AALST, W. VAN DER, MOLDT, D., VALK, R., WIENBERG, F., "Enacting Interorganizational Workflows using Nets in Nets", Proceedings of the 1999 Workflow Management Conference, Working Paper Series of the Department of Information Systems, vol. 70, pp. 117-136, University of Münster, Germany, November 1999

AALST, W. VAN DER, WESKE, M., "The P2P Approach to Interorganizational Workflows", in Dittrich, K. R., Geppert, A., Norrie, M. C. (Eds.), Proc. of the 13th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'01), Lecture Notes in Computer Science, vol. 2068, pp. 140-156, Springer-Verlag, 2001

ADOLPH, N., "Poor service? Download your rage", Financial Times, Connectis, Issue 5, pp. 39, October 2000

AFSARMANESH, H., GARITA, C., UGUR, Y., FRENKEL, A., HERTZBERGER, L., "Design of the Federated Information Management Architecture for PRODNET", pp. 127-146, in [Camarinha-Matos and Afsarmanesh, 1999a]

ALLEN, D., "Outsourcing as a Strategic Advantage", http://www.abacuspayment.com/docs/pdf_file/Outsourcing.pdf, 2002

ALONSO, G., AGRAWAL, D., ABBADI, A. EL, KAMATH, M., GÜNTHÖR, R., MOHAN, C., "Advanced Transaction Models in Workflow Contexts", Proceedings of the 12th Intl. Conference on Data Engineering, New Orleans, February 1996

ALONSO, G., AGRAWAL, D., ABBADI, A. EL, MOHAN, C., "Functionality and Limitations of Current Workflow Management Systems", IEEE Expert, vol. 12, no. 5, September-October 1997

AMICE ESPRIT CONSORTIUM, "CIMOSA: Open System Architecture for CIM", 2nd Ed., Springer-Verlag, 1993

ANDERSON, D., "SETI@home", in Oram, A. (Ed.), "Peer-to-Peer: Harnessing the Power of Disruptive Technologies", O'Reilly & Associates, 2001

ANDERSON, G., "From Supply Chain to Collaborative Commerce Networks: The Next Step in Supply Chain Management", Montgomery Research Inc, Achieving Supply Chain Excellence Through Technology (ASCET) vol. 2, pp. 101-105, 2000

ANTHONY, T., "Supply Chain Collaboration: Success in the New Internet Economy", Montgomery Research Inc, Achieving Supply Chain Excellence Through Technology (ASCET) vol. 2, pp. 41-44, 2000

ARIBA INC., "cXML User's Guide", version 1.2.007, http://www.cxml.org/, 2001

BAKER, A., PARUNAK, H., EROL, K., "Agents and the Internet: Infrastructure for Mass Customization", IEEE Internet Computing, vol. 3, no. 5, pp. 62-69, September/October 1999

BALBO, G., "Introduction to Stochastic Petri Nets", in Brinksma, E., Hermanns, H., Katoen, J. (Eds.), "Lectures on Formal Methods and Performance Analysis", Lecture Notes in Computer Science, vol. 2090, pp. 84-155, Springer-Verlag, 2001

BALLINGER, K., BRITTENHAM, P., MALHOTRA, A., NAGY, W., PHARIES, S., "Web Services Inspection Language (WS-Inspection) 1.0", http://www-106.ibm.com/developerworks/library/ws-wsilspec.html, 2001

BANAVAR, G., CHANDRA, T., STROM, R., STURMAN, D., "A Case for Message Oriented Middleware", Lecture Notes in Computer Science, 1693, Springer, 1999

BARKER, T., "Beyond the Grave", Financial Times, Connectis, Issue 5, pp. 42-45, October 2000

BATES, J., "Creating Lightweight Components with ATL", Sams Publishing, 1999

BECKER, J., UHR, W., VERING, O., "Retail Information Systems Based on SAP Products", Springer Verlag, 2001

BERGER, I., "Optimizing the Supply Chain with APS", APICS Magazine, December 1999

BERNERS-LEE, T., HENDLER, J., LASSILA, O., "The Semantic Web", Scientific American, May 2001

BERNERS-LEE, T., "The World Wide Web: Past, Present and Future", http://www.w3.org/People/Berners-Lee/1996/ppf.html, 1996

BERNUS, P., NEMES, L., WILLIAMS T., (Eds.), "Architectures for Enterprise Integration", Chapman & Hall, 1996

BERTINO, E., FERRARI, E., "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems", ACM Transactions on Information and System Security, vol. 2, no. 1, pp. 65-104, February 1999

BOLCER, G., "Magi: An Architecture for Mobile and Disconnected Workflow", IEEE Internet Computing, pp. 46-54, May/June 2000

BOLCER, G., KAISER, G., "SWAP: Leveraging the Web to Manage Workflow", IEEE Internet Computing, pp. 85-88, January/February 1999

BOLERO INTERNATIONAL LTD., "bolero.net: Accelerating global trade", http://www.bolero.net/, 1999

BOLERO INTERNATIONAL LTD., "boleroXML takes further steps towards full ebXML convergence", http://www.bolero.net/boleroxml/introduction/ebxml.php3, 1999

BOOCH, G. RUMBAUGH, J. JACOBSON, I., "The Unified Modeling Language User Guide", Addison-Wesley, 1999

BORGHOFF, U., SCHLICHTER, J., "Computer-Supported Cooperative Work: Introduction to Distributed Applications", Springer, 2000

BREMER, C., MUNDIM, A., MICHILINI, F., SIQUEIRA, J., ORTEGA, L., "A Brazilian Case of VE Coordination", pp. 377-386, in [Camarinha-Matos and Afsarmanesh, 1999a]

BRENNER, W., ZARNEKOW, R., WITTIG, H., "Intelligent Software Agents", Springer, 1998

BROOKS, R., "Intelligence without representation", Artificial Intelligence, no. 47, pp. 139-159, 1991

BROOS, R., DILLENSEGER, B., GUTHER, A., LEITH, M., "MIAMI: Mobile Intelligent Agents for Managing the Information Infrastructure", in "Agent Technology in Europe: ACTS Activities", InfoWin, ISBN 3-00-005267-4, available at http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/toc.htm, 1999

BROWN-HUMES, C., "Swedish model sets out to woo the rest of Europe", Connectis, Financial Times, Issue 5, pp. 29, October 2000

BUDIMAC, Z., IVANOVIC, M., POPOVIC, A., "Workflow Management System Using Mobile Agents", Advances in Databases and Information Systems, Lecture Notes in Computer Science, vol. 1691, pp. 168-178, Springer, 1999

CABRERA, F., COPELAND, G., COX, B., FREUND, T., KLEIN, J., STOREY, T., THATTE, S., "Web Services Transaction (WS-Transaction)", http://www-106.ibm.com/developerworks/library/ws-transpec/, 2002

CALDWELL, B., "Scan-Based Trading: Shared database to cut down on errors", Information Week Online, http://www.informationweek.com/725/vialink.htm, 1999

CAMARINHA-MATOS, L., AFSARMANESH, H. (Eds.), "Infrastructures for Virtual Enterprises: Networking Industrial Enterprises", Kluwer Academic Publishers, 1999

CAMARINHA-MATOS, L., AFSARMANESH, H., "The PRODNET Architecture", pp. 109-126, in [Camarinha-Matos and Afsarmanesh, 1999a]

CAMARINHA-MATOS, L., AFSARMANESH, H., "The Virtual Enterprise Concept", pp. 3-14, in [Camarinha-Matos and Afsarmanesh, 1999a]

CAMARINHA-MATOS, L., CARDOSO, T., "Selection of Partners for a Virtual Enterprise", pp. 259-278, in [Camarinha-Matos and Afsarmanesh, 1999a]

CAMARINHA-MATOS, L., LIMA, C., "PRODNET Coordination Module", pp. 147-166, in [Camarinha-Matos and Afsarmanesh, 1999a]

CAVANILLAS, S., NADAL, A., "Deliverable 2.1.7: Contract Law", ECLIP project, http://www.eclip.org/, 1999

CANE, A., "The mobile evolution", Understanding 3G, Financial Times, pp. 6-7, Summer 2001

CASE, S., AZARMI, N., THINT, M., OHTANI, T., "Enhancing E-Communities with Agent-Based Systems", IEEE Computer, vol. 34, no. 7, pp. 64-69, July 2001

CEN TC310, "Advanced Manufacturing Technology - Systems Architecture - Constructs for Enterprise Modelling", ENV 12204, 1996

CEN TC310 WG1, "Advanced Manufacturing Technology - Systems Architecture - Enterprise Model Execution and Integration Services", Version 3.3, CEN Report, 1998

CERAMI, E., "Web Service Essentials", O'Reilly & Associates, 2002

CERF, V., KAHN, R., "A Protocol for Packet Network Interconnection", IEEE Transactions on Communication Technology, Vol. COM-22, pp. 627-641, May 1974

CERN, "An overview of the World-Wide Web", http://public.web.cern.ch/Public/ACHIEVEMENTS/WEB/Welcome.html, 1997

CHANDRA, C., SMIRNOV, A., SHEREMETOV, L., "Agent-Based Infrastructure of Supply Chain Network Management", in Camarinha-Matos, L., Afsarmanesh, H., Rabelo, R. (Eds.), "E-Business and Virtual Enterprises: Managing Business-to-Business Cooperation", Kluwer Academic Publishers, 2000

CHAPPELL, D., "Understanding ActiveX and OLE", Microsoft Press, 1996

CHAUDHURI, S., DAYAL, U., "An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD Record, vol. 26, no. 1, March 1997

CHEN, Q., HSU, M., DAYAL, U., GRISS, M., "Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation", HP Laboratories, Technical Report HPL-1999-136, http://www.hpl.hp.com/techreports/1999/HPL-1999-136.html, 1999

CHESS, D., HARRISON, C., KERSHENBAUM, A., "Mobile Agents: Are They a Good Idea?", IBM Research Report, http://www.research.ibm.com/iagents/publications.html, 1994

CHRYSANTHIS, P., ZNATI, T., BANERJEE, S., CHANG, S.-K., "Establishing Virtual Enterprises by means of Mobile Agents", Proceedings of the 9th International Workshop on Research Issues on Data Engineering (RIDE-VE'99), Sydney, Australia, March 1999

CHUNG, P., HUANG, Y., YAJNIK, S., LIANG, D., SHIH, J., WANG, C.-Y., WANG, Y.-M., "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer", http://www.research.microsoft.com/~ymwang/papers/HTML/DCOMnCORBA/S.html, 1997

CLARK, D., "Face-to-Face with Peer-to-Peer Networking", IEEE Computer, vol. 34, no. 1, pp. 18-21, January 2001

CLARKE, I., SANDBERG, O., WILEY, B., HONG, T., "Freenet: A Distributed Anonymous Information Storage and Retrieval System", in Federrath, H. (Ed.), "Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability", Lecture Notes in Computer Science, vol. 2009, pp. 46-66, Springer, 2001

CLIP2 DISTRIBUTED SEARCH SOLUTIONS, "The Gnutella Protocol Specification v0.4", http://dss.clip2.com/, 2001

COLLINS, J., BILOT, C., GINI, M., MOBASHER, B., "Decision Processes in Agent-Based Automated Contracting", IEEE Internet Computing, vol. 5, no. 2, pp. 61-72, March/April 2001

COMMERCENET INC, "eCo Architecture for Electronic Commerce Interoperability", http://eco.commerce.net/rsrc/eCoSpec.pdf, 1999

COMMERCEROUTE INC, "Evaluating CommerceRoute(TM) 2.0 eBusiness Suite", http://www.commerceroute.com/pdfs/CR20TechWhitePaper.pdf, 2000

CORRY, C., MAYFIELD, V., CADMAN, J., MORIN, R., "The Waite Group's COM/DCOM Primer Plus", Sams Publishing, 1998

COUNSELL, A., "E-volution in Action", Financial Times, Connectis, Issue 5, pp. 8-16, October 2000

COUNSELL, A., "The 400 Blows", Financial Times, Connectis, Issue 10, pp. 22, April 2001

DABKE, P., "Enterprise Integration via CORBA-Based Information Agents", IEEE Internet Computing, vol. 3, no. 5, pp. 49-57, September/October 1999

DADAM, P., REICHERT, M., "The ADEPT WfMS Project at the University of Ulm", Proceedings of the 1st European Workshop on Workflow and Process Management (WPM'98), Zurich, Switzerland, October 1998

DALE, J., MAMDANI, E., "Open Standards for Interoperating Agent-Based Systems", Software Focus, vol. 2, no. 1, pp. 1-8, 2001

DAMODAR, Y., CAROL, L., "The just-in-time philosophy: A literature review", International Journal of Production Research, no. 29, vol. 4, pp. 657-676, 1991

DAN, A., DIAS, D., KEARNEY, R., LAU, T., NGUYEN, T., PARR, F., SACHS, M., SHAIKH, H., "Business-to-business integration with tpaML and a business-to-business protocol framework", IBM Systems Journal, vol. 40, no. 1, 2001

DAS, S., KOCHUT, K., MILLER, J., SHETH, A., WORAH, D., "ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR2", Technical Report UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, February 1997

DAVENPORT, T., "Mission Critical: Realizing the Promise of Enterprise Systems", Harvard Business School Press, 2000

DAVI, S., "Dying to make a difference", Financial Times, Connectis, Issue 8, pp. 43-45, February 2001

DAVID, R., ALLA, H., "Petri Nets and Grafcet: Tools for modelling discrete event systems", Prentice-Hall, 1992

DICKERHOF, M., DIDIC, M., MAMPEL, U., "Workflow and CIMOSA - background and case study", Computers in Industry, 40, pp. 197-205, 1999

DOUMEINGTS, G., VALLESPIR, B., DARRACAR, D., ROBOAM, M., "Design Methodology for Advanced Manufacturing Systems", Computers in Industry, vol. 9, no. 4, pp. 271-296, December 1987

DURFEE, E., "Scaling Up Agent Coordination Strategies", IEEE Computer, vol. 34, no. 7, pp. 39-46, July 2001

EASON, K., "Information Technology and Organisational Change", Taylor & Francis, 1988

EDWARDS, J., "3-Tier Client/Server At Work", John Wiley & Sons, 1999

EINSIEDLER, H., LÉGER, A., GLEIZES, M.-P., "ABROSE: A Co-operative Multi-Agent Based Framework for Electronic Marketplace", in "Agent Technology in Europe: ACTS Activities", InfoWin, ISBN 3-00-005267-4, available at http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/toc.htm, 1999

ENDL, R., KNOLMAYER, G., PFAHRER, M., "Modeling Processes and Workflows by Business Rules", Proceedings of the 1st European Workshop on Workflow and Process Management (WPM'98), Zurich, Switzerland, October 1998

FARMER, W., GUTTMAN, J., SWARUP, V., "Security for Mobile Agents: Issues and Requirements", Proceedings of the 19th National Information Systems Security Conference (NISSC '96), http://www.csrc.nist.gov/nissc/, Baltimore, USA, October 22-25, 1996

FERGUSON, I., "Touring Machines: Autonomous Agents with Attitudes", IEEE Computer, vol. 25, no. 5, pp. 51-55, May 1992

FERREIRA, D. (Ed.), "Modelo de Actividades da Empresa Piloto", Deliverable 1.1, PRONEGI Project, INESC Porto, 2000

FERREIRA, D. (Ed.), "Protótipo do Sistema", Deliverable 4.1, PRONEGI Project, INESC Porto, 2001

FERREIRA, D., FERREIRA, H., "Workflow Backbone: Pre-Release Version Instructions", DAMASCOS project, INESC Porto, January 2001

FERREIRA, D., FERREIRA, J. J. P., "Designing Workflow-Enabled Business-to-Business Infrastructures", Proceedings of the 7th International Conference on Concurrent Enterprising (ICE 2001), Bremen, Germany, 27th-29th June, 2001

FERREIRA, H., "Uma Arquitectura de Suporte à Integração Inter-Empresarial: infra-estrutura para a integração dos processos de negócio", MS Thesis, Faculdade de Engenharia da Universidade do Porto, 2001

FERREIRA, H., "Workflow Backbone: Uma infra-estrutura para implementar um sistema de workflow distribuído", Internal Report, INESC Porto, 1999

FERREIRA, J. J. P., "DAMASCOS: Dynamic Forecast for Master Production Planning with Stock and Capacity Constraints", Contract Preparation Form (CPF), Annex 1: Description of Work, INESC Porto, 1999

FERSCHA, A., "Qualitative and Quantitative Analysis of Business Workflows using Generalized Stochastic Petri Nets", in Proceedings of Workflow Management - Challenges Paradigms and Products (CON '94), pp. 222-234, Vienna, Austria, 1994

FININ, T., FRITZSON, R., MCKAY, D., MCENTIRE, R., "KQML as an Agent Communication Language", Proceedings of the Third International Conference on Information and Knowledge Management, pp. 456-463, ACM Press, 1994

FININ, T., LABROU, Y., MAYFIELD, J., "KIF101 - A brief introduction to the knowledge interchange format", http://www.cs.umbc.edu/kse/kif/kif101.shtml, 1995

FIPA (Foundation for Intelligent Physical Agents), "FIPA Abstract Architecture Specification", http://www.fipa.org/, 2000

FIPA (Foundation for Intelligent Physical Agents), "FIPA ACL Message Structure Specification", http://www.fipa.org/, 2000

FIPA (Foundation for Intelligent Physical Agents), "FIPA SL Content Language Specification", http://www.fipa.org/, 2000

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995

GAZZOTTI, D., "Web-linking Heterogeneous Applications for Large-scale Engineering and Services", 1st TRANSACT Cluster Workshop on Future Developments in Business Networks and Transaction Systems, Brussels, May 2001

GEORGAKOPOULOS, D., HORNICK, M., SHETH, A., "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, vol. 3, no. 2, pp. 119-153, April 1995

GEPPERT, A., KRADOLFER, M., TOMBROS, D., "Issues on Workflow Specification and Execution", Proceedings of the 1st European Workshop on Workflow and Process Management (WPM'98), Zurich, Switzerland, October 1998

GILLBLAD, D., HOLST, A., "Sales Prediction with Marketing Data Integration for Supply Chains", Proceedings of the 18th International Conference on CAD/CAM, Robotics and Factories of the Future (CARS&FOF 2002), Porto, Portugal, July 3-5, 2002

GLUSHKO, R., TENENBAUM, J., MELTZER, B., "An XML Framework for Agent-based E-commerce", Communications of the ACM, vol. 42, no. 3, pp. 106-114, March 1999

GRANT, J., "Safe as the Bank of Scotland", Connectis, Financial Times, Issue 7, pp. 23, December 2000

GREFEN, P., CERI, S., SÁNCHEZ, G., "Transaction and Rule Support for Workflow Management Systems - A Retrospective on the WIDE Architecture", Proceedings of the 1st European Workshop on Workflow and Process Management (WPM'98), Zurich, Switzerland, October 1998

GREFEN, P., ABERER, K., HOFFNER, Y., LUDWIG, H., "CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises", Intl. Journal of Computer Systems Science & Engineering, vol. 15, no. 5, pp. 277-290, 2000

GRISS, M., POUR, G., "Accelerating Development with Agent Components", IEEE Computer, vol. 34, no. 5, pp. 37-43, May 2001

GROISS, H., EDER, J., "Bringing Workflow Systems to the Web", http://www.ifi.uni-klu.ac.at/~herb/Overview.html, Institut für Informatik, Universität Klagenfurt, Austria, 1997

GROMOV, G., "The Roads and Crossroads of Internet History", http://www.netvalley.com/intval.html, 2000

GROVER, V., MALHOTRA, M., "Business process reengineering: A tutorial on the concept, evolution, method, technology and application", Journal of Operations Research, no. 15, vol. 3, pp. 193-213, 1997

HAGEN, C., ALONSO, G., "Flexible exception handling in the OPERA process support system", Proceedings of the 18th Intl. Conference on Distributed Computing Systems (ICDCS'98), Amsterdam, Netherlands, 1998

HALL, C., "What is VMI?", http://vendormanagedinventory.com/article5.htm, 1998

HANDEN, L., "The Tools for CRM: The Three Ws of Technology", in Brown, S. (Contrib. Ed.), "Customer Relationship Management: A Strategic Imperative in the World of e-Business", John Wiley & Sons, 2000

HANNEBAUER, M., "From Formal Workflow Models to Intelligent Agents", in Drabble, B., Jarvis, P. (Eds.), "Agent-Based Systems in the Business Context", Technical Report WS-99-02, AAAI Press, 1999

HASSAN, T., CARTER, C., HANNUS, M., HYVÄRINEN, J., "eLEGAL: Defining a Framework for Legally Admissible Use of ICT in Virtual Enterprises", Proceedings of the 7th International Conference on Concurrent Enterprising (ICE 2001), Bremen, Germany, 27th-29th June, 2001

HENDLER, J., "Is there an Intelligent Agent in Your Future?", Nature, http://www.nature.com/nature/webmatters/agents/agents.html, March 1999

HENNING, M., VINOSKI, S., "Advanced CORBA Programming with C++", Addison-Wesley, 1999

HILDRETH, S., "New Mapping Service Aims to Speed B2B Integration", http://b2b.ebizq.net/exchanges/hildreth_2.html, 2001

HOFFNER, Y. (Ed.), "Architecture Description", CrossFlow deliverable D3.a, CrossFlow Consortium / IBM Zurich Research Laboratory, http://www.crossflow.org/, 1999

HOLLINGER, P., "The Nightmare before Christmas", Financial Times, Connectis, Issue 7, pp. 10-18, December 2000

HUHNS, M., JACOBS, N., KSIEZYK, T., SHEN, W., SINGH, W., CANNATA, P., "Enterprise Information Modeling and Integration in Carnot", Enterprise Integration Modeling: Proceedings of the First International Conference, MIT Press, 1992

HUHNS, M., SINGH, M., "Distributed Artificial Intelligence for Information Systems", International Working Conference on Cooperating Knowledge Based Systems, University of Keele, UK, June 1994

HUHNS, M., SINGH, M., "Workflow Agents", IEEE Internet Computing, vol. 2, no.4, pp. 94-96, July/August 1998

IBM DEVELOPERWORKS, "Web services specifications", http://www-106.ibm.com/developerworks/webservices/library/ws-spec.html, 2002

IBM UNITED KINGDOM LTD, "Swedish bank enters mobile phone e-business area", e-business: a working reality, pp. 9, Issue 10, February 2000

IBM UNITED KINGDOM LTD, "New payment card boosts security for Web transactions", e-business: a working reality, pp. 6, Issue 13, January 2001

IBM UNITED KINGDOM LTD, "Electrical wholesaler customers welcome WAP enhancement", e-business: a working reality, pp. 14, Issue 13, January 2001

IBM UNITED KINGDOM LTD, "Danish mortgage company is first with WAP technology", e-business: a working reality, pp. 15, Issue 13, January 2001

IBM UNITED KINGDOM LTD, "Mobile computing revolutionises learning at the ThinkPad University", e-business: a working reality, pp. 17, Issue 13, January 2001

IBM UNITED KINGDOM LTD, "Election results service gets Belgian vote", e-business: a working reality, pp. 19, Issue 13, January 2001

IBM UNITED KINGDOM LTD, "IBM thrives on its own B2E cooking", e-business: a working reality, pp. 12, Issue 14, June 2001

IBM UNITED KINGDOM LTD, "Sales force automation delivers huge benefits", e-business: a working reality, pp. 13, Issue 14, June 2001

IETF, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, http://www.ietf.org/rfc/rfc2045.txt, 1996

IETF, "S/MIME Version 3 Message Specification", RFC 2633, http://www.ietf.org/rfc/rfc2633.txt, 1999

IFIP-IFAC TASK FORCE, "GERAM: Generalised Enterprise Reference Architecture and Methodology", Version 1.6.3, http://www.cit.gu.edu.au/~bernus/taskforce/geram/versions/geram1-6-3/v1.6.3.html, 1999

INAMOTO, A., "Agent oriented system approach for workflow automation", International Journal of Production Economics, vol. 60-61, pp. 327-335, April 1999

ISO/TC 176/SC 2, "ISO 9001:2000 Quality Management Systems - Requirements", Draft International Standard, 22 February 1999

IVEZIC, N., POTOK, T., POUCHARD, L., "Multiagent Framework for Lean Manufacturing", IEEE Internet Computing, vol. 3, no. 5, pp. 58-59, September/October, 1999

JABLONSKI, S., "MOBILE: A Modular Workflow Model and Architecture", Proceedings of the Intl. Conference on Dynamic Modelling and Information Systems, Noordwijkerhout, Netherlands, 1994

JAIN, A., APARICIO IV, M., SINGH, M., "Agents for Process Coherence in Virtual Enterprises", Communications of the ACM, vol. 42, no. 3, pp. 62-69, March 1999

JENNINGS, N., WOOLDRIDGE, M., "Agent-Oriented Software Engineering", Proceedings of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence, Cairo, Egypt, 1999

JOSHI, J., AREF, W., GHAFOOR, A., SPAFFORD, E., "Security Models for Web-based Applications", Communications of the ACM, vol. 44, no. 2, pp. 38-44, February 2001

JUDGE, D., ODGERS, B., SHEPHERDSON, J., CUI, Z., "Agent-enhanced workflow", BT Technology Journal, vol. 16, no. 3, July 1998

KAISER, G., STONE, A., DOSSICK, S., "A Mobile Agent Approach to Lightweight Process Workflow", Proceedings of the International Process Technology Workshop (IPTW), http://www-adele.imag.fr/IPTW/, Grenoble, France, September 1-3, 1999

KAK, R., SCHOONMAKER, M., "RosettaNet E-Business Standards Provide Better Supply Chain Collaboration and Efficiency", Montgomery Research Inc, Achieving Supply Chain Excellence Through Technology (ASCET) vol. 4, 2002

KALAKOTA, R., WHINSTON, A., "Electronic Commerce: A Manager's Guide", Addison-Wesley, New York, 1996

KALER, C. (Ed.), "Web Services Security (WS-Security)", http://www-106.ibm.com/developerworks/library/ws-secure/, 2002

KALER, C. (Ed.), "WS-Security Profile for XML-based Tokens", http://www-106.ibm.com/developerworks/library/ws-sectoken.html, 2002

KAPPEL, G., PRÖLL, B., RAUSCH-SCHOTT, S., RETSCHITZEGGER, W., "TriGSflow: Active Object-Oriented Workflow Management", Proceedings of the 28th Hawaii Intl. Conference on System Sciences, Hawaii, USA, 1995

KIM, Y., KANG, S.-H., KIM, D., BAE, J., JU, K.-J., "WW-FLOW: Web-Based Workflow Management with Runtime Encapsulation", IEEE Internet Computing, pp. 55-64, May/June 2000

KIMBALL, D., KIMBALL JR., D., "Principles of Industrial Organization", Sixth Edition, McGraw-Hill, 1947

KLEINROCK, L., "Information Flow in Large Communication Nets", RLE Quarterly Progress Report, July 1961

KLEN, A., RABELO, R., SPINOSA, L., FERREIRA, A., "Distributed Business Process Management", pp. 241-258, in [Camarinha-Matos and Afsarmanesh, 1999a]

KLINGEMANN, J., WÄSCH, J., ABERER, K., "Deriving Service Models in Cross-Organizational Workflows", Proceedings of the 9th International Workshop on Research Issues on Data Engineering (RIDE-VE'99), Sydney, Australia, March 1999

KNOLMAYER, G., MERTENS, P., ZEIER, A., "Supply Chain Management Based on SAP Systems: Order Management in Manufacturing Companies", Springer Verlag, 2001

KONICKI, "GM Buys $96 Billion In Materials Via Covisint", InformationWeek, http://www.informationweek.com/story/IWK20010816S0001, August 16, 2001

KORPELA, E., WERTHIMER, D., ANDERSON, D., COBB, J., LEBOFSKY, M., "SETI@home: Massively Distributed Computing for SETI", IEEE Computing in Science and Engineering, Vol. 3, No. 1, 2001

KRAEMER, K., DEDRICK, J., "Strategic use of the Internet and e-commerce: Cisco Systems", Journal of Strategic Information Systems, no. 11, pp. 5-29, 2002

KRONZ, A., "Einführung von Workflow-Systemen mit ARIS Modellen", in Scheer, A.-W., "ARIS - Modellierungsmethoden, Metamodelle, Anwendungen", Springer Verlag, 2001

KTENIDIS, P., PARASKEVOPOULOS, A., "Information and Communication Technology (ICT) Infrastructure for Small and Medium Enterprises (SMEs) Networks: Achievements and Work-in-Progress from Three European Projects", Proceedings of the 9th International Workshop on Research Issues on Data Engineering (RIDE-VE'99), Sydney, Australia, March 1999

KÜHN, W., "Earn as you learn", Financial Times, Connectis, Issue 13, pp. 18-19, July 2001

LABROU, Y., FININ, T., "A Proposal for a new KQML Specification", TR CS-97-03, Computer Science and Electrical Engineering Department, University of Baltimore Maryland County, http://www.cs.umbc.edu/, February 1997

LAMBERT, D., COOPER, M., PAGH, J., "Supply Chain Management: Implementation Issues and Research Opportunities", International Journal of Logistics Management, vol.9, no.2, pp. 1-19, 1998

LAMBERT, D., STOCK, J., ELLRAM, L., "Fundamentals of Logistics Management", McGraw-Hill International Editions, 1998

LAMOND, K., EDELHEIT, J., "Electronic commerce back-office integration", BT Technology Journal, vol. 17, no. 3, pp. 87-96, July 1999

LANCESSEUR, B., "Baptism of Fire", Financial Times, Connectis, Issue 7, pp. 38-42, December 2000

LANGE, D., OSHIMA, M., "Programming and Deploying Java(TM) Mobile Agents with Aglets(TM)", Addison-Wesley, 1998

LANGE, D., OSHIMA, M., "Seven Good Reasons for Mobile Agents", Communications of the ACM, vol. 42, no. 3, pp. 88-89, March 1999

LAUBE, H., "Revenge of the Computer Anarchists", Financial Times, Connectis, Issue 8, pp. 10-18, February 2001

LAWRENCE, P., "Workflow Handbook 1997", John Wiley & Sons, 1997

LAZCANO, A., ALONSO, G., SCHULDT, H., SCHULER, C., "The WISE Approach to Electronic Commerce", International Journal of Computer Systems Science & Engineering, vol. 15, no. 5, September 2000

LEINER, B., CERF, V., CLARK, D., KAHN, R., KLEINROCK, L., LYNCH, D., POSTEL, J., ROBERTS, L., WOLFF, S., "A Brief History of the Internet", http://www.isoc.org/internet-history/brief.html, 2000

LESCHER, F., "Making Your Enterprise Internet-Ready: E-Business for the Process Industries", Montgomery Research Inc, Achieving Supply Chain Excellence Through Technology (ASCET) vol. 2, 2000

LINTHICUM, D., "Process Automation and EAI", eAI Journal, http://www.eaijournal.com/, March 2000

MACEDO, B., FERREIRA, J. J. P., "Suporte à Integração de Processos de Negócio, Aplicação aos Processos de Gestão da Qualidade Total", Proposta de Projecto de Investigação em Consórcio, INESC Porto, 1998

MAES, P., "Agents that Reduce Work and Information Overload", Communications of the ACM, vol. 37, no.7, pp. 31-40, July 1994

MALONE, T., CROWSTON, K., "The Interdisciplinary Study of Coordination", ACM Computing Surveys, vol. 26, no. 1, pp. 87-119, 1994

MANGINA, E., "Review of Software Products for Multi-Agent Systems", AgentLink, http://www.agentlink.org/, June 2002

MARIER, S., MHAMEDI, A. EL, BINDER, Z., "Analysis of a Computer-Aided Teleoperation Process by means of Generalized Stochastic Petri Nets", Control Engineering Practice, vol. 5, no. 7, pp. 931-942, 1997

MARTINS, R., CHAVES, M., PIRMEZ, L., CARMO, L., "Mobile agent applications", Internet Research: Electronic Networking Applications and Policy, vol. 11, no. 1, pp. 59-54, 2001

MAURER, F., DELLEN, B., BENDECK, F., GOLDMANN, S., HOLZ, H., KÖTTING, B., SCHAAF, M., "Merging Project Planning and Web-Enabled Dynamic Workflow Technologies", IEEE Internet Computing, pp. 65-74, May/June 2000

MCILRAITH, S., SON, T., ZENG, H., "Semantic Web Services", IEEE Intelligent Systems, vol. 16, no. 2, pp. 46-53, March/April 2001

MEDEIROS, C., VOSSEN, G., WESKE, M., "WASA: A Workflow-Based Architecture to Support Scientific Database Applications", Proceedings of the 6th DEXA Conference, Lecture Notes in Computer Science, vol. 978, pp. 574-583, Springer, 1995

MEDINA-MORA, R., WONG, H., FLORES, P., "ActionWorkflow as the Enterprise Integration Technology", Data Engineering, IEEE Computer Society Press, vol. 16, no. 2, pp. 49-52, 1993

MENG, J., HELAL, S., SU, S., "An Ad-Hoc Workflow System Architecture Based on Mobile Agents and Rule-Based Processing", Proceedings of the International Conference on Parallel and Distributed Computing Techniques and Applications, Las Vegas, Nevada, USA, June 2000

MERLAT, W., "An agent-based multiservice negotiation for eCommerce", BT Technology Journal, vol. 17, no. 4, pp. 168-175, October 1999

MERTENS, P., GRIESE, J., EHRENBERG, D. (Eds.), "Virtuelle Unternehmen und Informationsverarbeitung", Springer, 1998

MERZ, M., GRIFFEL, F., TU, T., MÜLLER-WILKEN, S., WEINREICH, H., BOGER, M., LAMERSDORF, W., "Supporting Electronic Commerce Transactions with Contracting Services", Intl. Journal of Cooperative Information Systems, vol. 7, no. 4, pp. 249-274, 1998

MIAMI CONSORTIUM, "Major Event 4: Integrated MIAMI Demonstration", http://www.fokus.gmd.de/research/cc/ecco/miami/, 2000

MICROSOFT, DEC, "The Component Object Model Specification", Version 0.9, http://www.microsoft.com/com/resources/comdocs.asp, 1995

MICROSOFT, "BizTalk Framework 2.0: Document and Message Specification", http://www.biztalk.org/, 2000

MICROSOFT, "BizTalk Accelerator for RosettaNet", http://www.microsoft.com/biztalk/evaluation/rosettanet/, 2002

MILLER, J., SHETH, A., KOCHUT, K., WANG, X., "CORBA-Based Run-Time Architectures for Workflow Management Systems", Journal of Database Management, vol. 7, no. 1, pp. 16-27, 1996

MILLER, J., PALANISWAMI, D., SHETH, A., KOCHUT, K., SINGH, H., "WebWork: METEOR2's Web-Based Workflow Management System", Journal of Intelligent Information Systems, vol. 10, no. 2, pp. 185-215, 1998

MILOJICIC, D., BREUGST, M., BUSSE, I., CAMPBELL, J., COVACI, S., FRIEDMAN, B., KOSAKA, K., LANGE, D., ONO, K., OSHIMA, M., THAM, C., VIRDHAGRISWARAN, S., WHITE, J., "MASIF: The OMG Mobile Agent System Interoperability Facility", in Rothermel, K., Hohl, F. (Eds.), "Mobile Agents", Lecture Notes in Computer Science, 1477, Springer, 1998

MINAR, N., HEDLUND, M., "A Network of Peers: Peer-to-Peer Models Through the History of the Internet", in Oram, A. (Ed.), "Peer-to-Peer: Harnessing the Power of Disruptive Technologies", O'Reilly & Associates, 2001

MOHAN, C., AGRAWAL, D., ALONSO, G., ABBADI, A., GÜNTHÖR, R., KAMATH, M., "Exotica: A Project on Advanced Transaction Management and Workflow Systems", ACM SIGOIS Bulletin, vol. 16, no. 1, pp. 45-50, August 1995

MOLLER, F., STEVENS, P., "The Edinburgh Concurrency Workbench (Version 7.1)", Laboratory for Foundations of Computer Science, University of Edinburgh, 1997

MOLLOY, M., "Performance Analysis Using Stochastic Petri Nets", IEEE Transactions on Computers, vol. C-31, no. 9, pp. 913-917, September 1982

MRO SOFTWARE INC, "Conquering the Catalogue Challenge", http://www.mro.com/corporate/pdf/Catalog_Management_Whitepaper.pdf, 2002

MÜLLER, B., STOLP, P., "Workflow Management in der industriellen Praxis: Vom Buzzword zum High-Tech-Instrument", Springer, 1999

MÜLLER, J., PISCHEL, M., "The Agent Architecture InteRRaP: Concept and Application", DFKI Research Report RR-93-26, http://www.dfki.uni-kl.de/dfkidok/publications/RR/93/26/abstract.html, 1993

MÜLLER-WILKEN, S., "XML and Jini - On Using XML and the Java Border Service Architecture to Integrate Mobile Devices into the Java Intelligent Network Infrastructure", Proceedings of the XML Developers Conference (XTECH 2000), San Jose, California, USA, 2000

MUTH, P., WEISSENFELS, J., GILLMANN, M., WEIKUM, G., "Mentor-lite: Integrating Light-Weight Workflow Management Systems with Business Environments", Proceedings of the 1st European Workshop on Workflow and Process Management (WPM'98), Zurich, Switzerland, October 1998

NAIRN, G., "What went wrong with WAP?", Financial Times, Connectis, Issue 12, pp. 10-11, June 2001

NETSCAPE COMMUNICATIONS CORPORATION, "Netscape, Sun and Hewlett-Packard Join to Support New Internet-Based Standard for Workflow", http://wp.netscape.com/newsref/pr/newsrelease597.html, 1998

NEUMAN, B., TS'O, T., "Kerberos: An Authentication Service for Computer Networks", IEEE Communications, vol. 32, no. 9, pp. 33-38, 1994

NIEDERCORN, F., "Education overcomes the tyranny of distance", Financial Times, Connectis, Issue 5, pp. 54-56, October 2000

NIELSEN, H., CHRISTENSEN, E., FARRELL, J., "WS-Attachments", http://www-106.ibm.com/developerworks/library/ws-attach.html, 2002

NIELSEN, H., SANDERS, H., BUTEK, R., NASH, S., "Direct Internet Message Encapsulation (DIME)", http://www-106.ibm.com/developerworks/library/ws-dime/, 2002

NORRIS, M., WEST, S., "eBusiness Essentials: Technology and Network Requirements for Mobile and Online Markets", John Wiley & Sons, 2001

NWANA, H. LEE, L., JENNINGS, N., "Co-ordination in software agent systems", BT Technology Journal, vol. 14, no. 4, pp. 79-88, October 1996

NWANA, H., NDUMU, D., "An introduction to agent technology", BT Technology Journal, vol. 14, no. 4, pp. 55-67, October 1996

OAKLAND, J., "Total Quality Management", Butterworth-Heinemann, 1993

OASIS, "UDDI Version 3.0, Published Specification", http://www.uddi.org/, 2002

OBI CONSORTIUM, "Open Buying on the Internet (OBI) Technical Specifications", version 3.0, http://www.openbuy.org/, 2001

O'BRIEN, P., NICOL, R., "FIPA - towards a standard for software agents", BT Technology Journal, vol. 16, no. 3, pp. 51-59, July 1998

OMG, "MASIF-RTF Results", http://cgi.omg.org/docs/orbos/98-03-09.pdf, 1998

OMG, "Workflow Management Facility Specification", version 1.2, http://www.omg.org/cgi-bin/doc?formal/00-05-02.pdf, 2000

OMG, "Notification Service Specification", version 1.0, http://www.omg.org/cgi-bin/doc?formal/00-06-20.pdf, 2000

OMG, "Trading Object Service Specification", version 1.0, http://www.omg.org/cgi-bin/doc?formal/00-06-27.pdf, 2000

OMG, "Event Service Specification", version 1.1, http://www.omg.org/cgi-bin/doc?formal/01-03-01, 2001

O'NEILL, P., SOHAL, A., "Business Process Reengineering: A review of recent literature", Technovation, 19, pp. 571-581, 1999

ORTIZ, A., LARIO, F., ROS, L., HAWA, M., "Building a production planning process with an approach based on CIMOSA and workflow management systems", Computers in Industry, 40, pp. 207-219, 1999

OSÓRIO, A., ANTUNES, C., BARATA, M., "The PRODNET Communication Infrastructure", pp. 167-186, in [Camarinha-Matos and Afsarmanesh, 1999a]

ÖSTERLE, H., FLEISCH, E., ALT, R., "Business Networking: Shaping Collaboration Between Enterprises", Springer, 2001

OUZOUNIS, E., "An Agent-Based Platform for the Management of Dynamic Virtual Enterprises", PhD Thesis, Technical University of Berlin, 2001

PANCERELLA, C., BERRY, N., "Adding Intelligent Agents to Existing EI Frameworks", IEEE Internet Computing, vol. 3, no. 5, pp. 60-61, September/October 1999

PAPADOPOULOS, G., ARBAB, F., "Coordination Models and Languages", in Zelkowitz, M. (Ed.), "Advances in Computers", vol. 46, Academic Press, 1998

PARR, F. (Ed.), "HTTPR Specification", http://www-106.ibm.com/developerworks/library/ws-httprspec/, 2002

PARUNAK, H., "What can Agents do in Industry, and Why? An Overview of Industrially-Oriented R&D at CEC", in Klusch, M., Weiss, G. (Eds.), Cooperative Information Agents II, Lecture Notes in Computer Science, vol. 1435, Springer, 1998

PARUNAK, H., "Practical and Industrial Applications of Agent-Based Systems", http://www.erim.org/~vparunak/apps98.pdf, 1998

PEREIRA, J. (Ed.), "Actividades Padrão e Operações Funcionais", Deliverable 1.3, PRONEGI Project, INESC Porto, 2000

PETRIE, C., SARIN, S., "Beyond Documents: Sharing Work", IEEE Internet Computing, vol. 4, no. 3, pp. 34-36, May/June 2000

PICCINELLI, G., STEFANELLI, C., MORCINIEC, M., CASASSA-MONT, M., "Policy-based Management for E-Service Delivery", HP OpenView University Association (HP-OVUA) 8th Annual Workshop, Berlin, June 24-27, 2001

PIF WORKING GROUP, "The PIF Process Interchange Format and Framework Version 1.1", Working Paper 194, MIT Center for Coordination Science, 1996

PONTRANDOLFO, P., OKOGBAA, O. G., "Global Manufacturing: a review and a framework for planning in a global corporation", International Journal of Production Research, vol. 37, no. 1, pp. 1-19, 1999

PUNZAK, J., "Oracle Internet Procurement", http://www.dlt.com/oracletechday/pres/Punzak.pdf, 2001

RABELO, R., KLEN, A., "Business Intelligence Support for Supply Chain Management", in Marik, V., Camarinha-Matos, L., Afsarmanesh, H. (Eds.), "Knowledge and Technology Integration in Production and Services", Kluwer Academic Publishers, pp. 437-444, 2002

RAJ, G., "A Detailed Comparison of CORBA, DCOM and Java/RMI", http://my.execpc.com/~gopalan/misc/compare.html, 1998

RANKY, P., "Computer-Integrated Manufacturing", Prentice-Hall, 1986

RATNASINGAM, P., "Inter-organizational trust in EDI adoption: the case of Ford Motor Company and PBR Limited in Australia", Internet Research: Networking Applications and Policy, Vol. 11, No. 3, pp. 261-268, 2001

RECTOR, B., SELLS, C., "ATL Internals", Addison-Wesley, 1999

REICHERT, M., HENSINGER, C., DADAM, P., "Supporting Adaptive Workflows in Advanced Application Environments", Proceedings of the EDBT Workshop on Workflow Management Systems, pp. 100-109, Valencia, Spain, March 1998

REICHERT, M., BAUER, T., DADAM, P., "Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows", in Dadam, P., Reichert, M. (Eds.), "Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications", Proceedings of Workshop Informatik'99, Paderborn, Germany, October 1999

RICCI, A., OMICINI, A., DENTI, E., "The TuCSoN Coordination Infrastructure for Virtual Enterprises", Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises Workshop, IEEE 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2001), MIT, Cambridge, Massachusetts, USA, June 2001

RIPEANU, M., IAMNITCHI, A., FOSTER, I., "Mapping the Gnutella Network", IEEE Internet Computing, vol. 6, no. 1, pp. 50-57, January/February 2002

RITTER, J., "Why Gnutella Can't Scale. No, Really.", http://www.darkridge.com/~jpr5/doc/gnutella.html, 2001

ROCHA, A., OLIVEIRA, E., "An Electronic Market Architecture for the Formation of Virtual Enterprises", in [Camarinha-Matos and Afsarmanesh, 1999a]

RODRIGUES, P., LOPES, F., "Salsa Server Applications", SALSA software manuals, ParaRede ICT, 2002

RODRIGUES, P., LOPES, F., "Salsa Server Install", SALSA software manuals, ParaRede ICT, 2002

RODRÍGUEZ, J., "Card-carrying Catastrophe", Financial Times, Connectis, Issue 10, pp. 38-40, April 2001

ROSETTANET, "Overview of Clusters, Segments and PIPs", http://www.rosettanet.org/, 2000

ROSETTANET, "RosettaNet Implementation Framework: Core Specification", version 02.00.01, http://www.rosettanet.org/, 2002

ROSETTANET, "Trading Partner Agreements v02.00 Release", http://www.rosettanet.org/tradingpartneragreements/, 2002

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F., LORENSON, W., "Object-Oriented Modeling and Design", Prentice Hall, 1990

SACHS, M., DAN, A., NGUYEN, T., KEARNEY, R., SHAIKH, H., DIAS, D., "Executable Trading-Partner Agreements in Electronic Commerce", IBM T. J. Watson Research Center, January 2000

SACHS, M., IBBOTSON, J., "Electronic Trading-Partner Agreement for E-Commerce (tpaML)", version 1.0.6, IBM Corporation, 2000

SAFFADY, W., "Electronic Document Imaging Systems: Design, Evaluation, and Implementation", Meckler, 1993

SAINT-BLANCAT, J. (Ed.), "Final Report", CrossFlow deliverable D16, CrossFlow Consortium, http://www.crossflow.org/, 2000

SANDHOLM, T., LESSER, V., "Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework", in Huhns, M., Singh, M. (Eds.), "Readings in Agents", Morgan Kaufmann, 1998

SANDHU, R., COYNE, E., FEINSTEIN, H., YOUMAN, C., "Role-Based Access Control Models", IEEE Computer, vol. 29, no. 2, pp. 38-47, February 1996

SCHEER, A.-W., "Architecture for Integrated Manufacturing Systems", Springer-Verlag, 1992

SCHEER, A.-W., HABERMANN, F., "Making ERP a Success", Communications of the ACM, vol. 43, no. 4, pp. 57-61, 2000

SCHMIDT, M.-T., "The Evolution of Workflow Standards", IEEE Concurrency, pp. 44-52, July-September 1999

SCHODER, D., EYMANN, T., "The Real Challenges of Mobile Agents", Communications of the ACM, vol. 43, no. 6, pp. 111-112, June 2000

SCHÖLKOPF, B., BURGES, C., MIKA, S., "Introduction to Support Vector Learning", in Schölkopf, B., Burges, C., Mika, S. (Eds.), "Advances in Kernel Methods: Support Vector Learning", MIT Press, 1998

SCHOONMAKER, M., "RosettaNet Standards: Strengthening Trading Networks", Montgomery Research Inc, Achieving Supply Chain Excellence Through Technology (ASCET) vol. 3, 2001

SCHULENBURG, W., "Workflow und anderes - eine handbare Übersicht", in Köhler-Frost, W. (Ed.), "Electronic Office Systeme: Workflow-Anwendungen und Groupware-Anwendungen in der Praxis", Erich Schmidt Verlag, 1998

SCHULER, C., SCHULDT, H., ALONSO, G., SCHEK, H.-J., "Workflow over Workflows: Practical Experiences with the Integration of SAP R/3 Business Workflow in WISE", in Dadam, P., Reichert, M. (Eds.), "Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications", Proceedings of Workshop Informatik'99, Paderborn, Germany, October 1999

SCHULZE, W., BÖHM, M., "Klassifikation von Vorgangsverwaltungssystemen", in Vossen, G., Becker, J., "Geschäftsprozess-modellierung und Workflow Management: Modelle, Methoden, Werkzeuge", Intl. Thomson Publishing, 1996

SCHUMACHER, M., CHANTEMARGUE, F., SCHUBIGER, S., HIRSBRUNNER, B., KRONE, O., "The STL++ Coordination Language: Application to Simulating the Automation of a Trading System", Proceedings of the First International Conference on Enterprise Information Systems, pp. 292-299, Setúbal, Portugal, March 1999

SCHWENKREIS, F., "APRICOTS - A Prototype Implementation of a ConTract System: Management of the Control Flow and the Communication System", Proceedings of the 12th Symposium on Reliable Distributed Systems, pp. 12-21, IEEE Computer Society Press, 1993

SHAPIRO, J., "Modeling the Supply Chain", Duxbury Press, 2000

SHEGALOV, G., GILLMANN, M., WEIKUM, G., "XML-enabled Workflow Management for E-Services across Heterogeneous Platforms", VLDB Journal, vol. 10, no. 1, pp. 91-103, 2001

SHEN, W., NORRIE, D., "An Agent-Based Approach for Manufacturing Enterprise Integration and Supply Chain Management", Proceedings of the Tenth International IFIP TC5 WG-5.2 WG-5.3 Conference PROLAMAT '98 (The Globalization of Manufacturing in the Digital Communications Era of the 21st Century: Innovation, Agility, and the Virtual Enterprise), Trento, Italy, September 9-11, 1998

SHEN, W., NORRIE, D., "Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey", Knowledge and Information Systems: An International Journal, vol. 1, no. 2, pp. 129-156, May 1999

SHEN, W., NORRIE, D., "Implementing Internet Enabled Virtual Enterprises Using Collaborative Agents", in [Camarinha-Matos and Afsarmanesh, 1999a]

SHEPHERDSON, J., THOMPSON, S., ODGERS, B., "Cross Organisational Workflow Co-ordinated by Software Agents", Cross-Organisational Workflow Management Workshop, International Joint Conference on Work Activities Coordination and Collaboration (WACC'99), San Francisco, California, USA, February 22-25, 1999

SHEPHERDSON, J., THOMPSON, S., ODGERS, B., "Decentralised workflows and software agents", BT Technology Journal, vol. 17, no. 4, pp. 65-71, October 1999

SHETH, A., AALST, W. VAN DER, ARPINAR, I., "Processes Driving the Networked Economy", IEEE Concurrency, pp. 18-31, July-September 1999

SHETH, A., RUSINKIEWICZ, M., "On Transactional Workflows", Data Engineering, IEEE Computer Society Press, vol. 16, no. 2, pp. 37-40, 1993

SIDLER, G., SCOTT, A., WOLF, H., "Collaborative Browsing in the World Wide Web", Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland, 1997

SLIM TECHNOLOGIES LLC, "Summary of SLIM/2000 Features", http://www.slimcorp.com/detailed.pdf, 2002

A. Smith, "An Inquiry into the Nature and Causes of the Wealth of Nations", 1776

SOTTO, R., "The Virtual Organisation", Accting. Mgmt. & Info. Tech., vol. 7, no. 1, pp. 37-51, 1997

SPROULE, S., ARCHER, N., "A buyer behaviour framework for the development and design of software agents in e-commerce", Internet Research: Electronic Networking Applications and Policy, vol. 10, no. 5, pp. 396-405, 2000

STAFFWARE PLC, "The Staffware Technical Overview", http://www.staffware.com/, 1999

STRICKER, C., RIBONI, S., KRADOLFER, M., TAYLOR, J., "Market-based Workflow Management for Supply Chain of Services", Proceedings of the 33rd Hawaii Intl. Conference on System Sciences (HICSS-33), Hawaii, January 2000

SUN MICROSYSTEMS, "Java Transaction API", version 1.0.1, http://www.java.sun.com/products/jta/docs.html, 1999

SUN MICROSYSTEMS, "Java Message Service", version 1.1, http://www.java.sun.com/products/jms/docs.html, 2002

SWENSON, K., "Netscape Explains SWAP Release", http://www.scripting.com/98/04/stories/netscapeExplainsSwap.html, 1998

SWENSON, K., "Simple Workflow Access Protocol (SWAP)", http://www.globecom.net/ietf/draft/draft-swenson-swap-prot-00.html, 1998

SYCARA, K., "The Many Faces of Agents", AI Magazine, vol. 19, no. 2, pp. 11-12, 1998

SYCARA, K., "Multiagent Systems", AI Magazine, vol. 19, no. 2, pp. 79-92, 1998

TALLMAN, O., KAIN, J., "COM versus CORBA: A Decision Framework", http://www.quoininc.com/company/articles/COM_CORBA.pdf, 1998

THATTE, S., "Business Process Execution Language for Web Services, Version 1.0", http://www-106.ibm.com/developerworks/library/ws-bpel/, 2002

THOMAS, A., "Java 2 Platform, Enterprise Edition: Ensuring Consistency, Portability, and Interoperability", Patricia Seybold Group, June 1999

TIMMERS, P., "Electronic Commerce: strategies and models for business-to-business trading", John Wiley & Sons, 1999

TOLKSDORF, R., "Models of Coordination", in Omicini, A., Tolksdorf, R., Zambonelli, F. (Eds.), "Engineering Societies in the Agents World", Lecture Notes in Artificial Intelligence, vol. 1972, pp. 78-92, Springer-Verlag, December 2000

TOMLINSON, C., ATTIE, P., CANNATA, P., MEREDITH, G., SHETH, A., SINGH, M., WOELK, D., "Workflow Support in Carnot", Data Engineering, IEEE Computer Society Press, vol. 16, no. 2, pp. 33-36, 1993

TOTLAND, T., CONRADI, R., "A Survey and Classification of Some Research Areas Relevant to Software Process Modeling", Proceedings of the 4th European Workshop on Software Process Technology, Noordwijkerhout, Netherlands, 1995

TREUHAFT, J., "Overview of SSL 3.0", Netscape Internet Developer Conference, http://developer.netscape.com/misc/developer/conference/proceedings/cs2/index.html, 1996

UN/CEFACT, OASIS, "ebXML Business Process Specification Schema", version 1.01, http://www.ebxml.org/, 2001

UN/CEFACT, OASIS, "ebXML Collaboration-Protocol Profile and Agreement Specification", version 1.0, http://www.ebxml.org/, 2001

UN/CEFACT, OASIS, "ebXML Registry Information Model", version 2.0, http://www.ebxml.org/, 2001

UN/CEFACT, OASIS, "ebXML Registry Services Specification", version 2.0, http://www.ebxml.org/, 2001

UN/CEFACT, OASIS, "ebXML Technical Architecture Specification", version 1.0.4, http://www.ebxml.org/, 2001

UN/CEFACT, OASIS, "ebXML Message Service Specification", version 2.0, http://www.ebxml.org/, 2002

UNITT, M., JONES, I. C., "EDI - the grand daddy of electronic commerce", BT Technology Journal, Vol. 17, No. 3, pp. 17-23, July 1999

USCHOLD, M., GRUNINGER, M., "Ontologies: principles, methods, and applications", Knowledge Engineering Review, vol. 11, no. 2, pp. 93-155, 1996

VEERAMANI, R., TALBERT, N., "Looking Back at Struggles, Looking Ahead to Opportunities", IEEE IT Pro, pp. 15-17, Jan./Feb. 2001

VENNERS, B., "Inside the Java Virtual Machine", McGraw-Hill Osborne Media, 2000

VERNADAT, F., "Enterprise Modelling and Integration: Principles and Applications", Chapman & Hall, 1996

W3C, "Extensible Markup Language", http://www.w3.org/XML/, 1997

W3C, "Namespaces in XML", http://www.w3.org/TR/1999/REC-xml-names-19990114/, 1999

W3C, "XSL Transformations Version 1.0", W3C Recommendation, http://www.w3.org/TR/xslt, 1999

W3C, "A Little History of the World Wide Web", http://www.w3.org/History.html, 2000

W3C, "Simple Object Access Protocol (SOAP) 1.1", http://www.w3.org/TR/SOAP/, 2000

W3C, "XML Schema Specifications and Development", http://www.w3.org/XML/Schema#dev, 2000

W3C, "Web Services Description Language (WSDL) 1.1", http://www.w3.org/TR/wsdl/, 2001

W3C, "XML-Signature Syntax and Processing", W3C Recommendation, http://www.w3.org/TR/xmldsig-core, 2002

WÄCHTER, H., REUTER, A., "The ConTract Model", in Elmagarmid, A. (Ed.), "Advanced Transaction Models for New Applications", Morgan Kaufmann Publishers, 1992

WANG, A., "Using software agents to support evolution of distributed workflow models", Proceedings of the International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000), International ICSC Congress on Intelligent Systems and Applications (ISA'2000), Wollongong, Australia, December 12-15, 2000

WATSON III, J., DESROCHERS, A., "Applying Generalized Stochastic Petri Nets to Manufacturing Systems Containing Nonexponential Transition Functions", IEEE Transactions on Systems Man and Cybernetics, vol. 21, no. 5, pp. 1008-1016, Sep/Oct 1991

WATSON, J. P., DIAZ, E., SZKATULA, K., "Strategic Analysis of the Innovation Relay Centre (IRC) Network: Executive Summary", Directorate C - Innovation, Unit C3 Innovation Networks and Services, European Commission, 2001

WEINBERG, F., "COSMOS: An Electronic Contracting Service Platform Enacting Contract Execution", CrossFlow Workshop (http://www.crossflow.org/), Rüschlikon, Switzerland, September 2000

WESKE, M., VOSSEN, G., "The WASA Project: A Survey", Proceedings of the 1st European Workshop on Workflow and Process Management (WPM'98), Zurich, Switzerland, October 1998

WEIß, P., KÖLMEL, B., "Customer Relationship Management and Smart Organization", Proceedings of the 18th International Conference on CAD/CAM, Robotics and Factories of the Future (CARS&FOF 2002), Porto, Portugal, July 3-5, 2002

WEISSENFELS, J., MUTH, P., WEIKUM, G., "Flexible Worklist Management in a Light-Weight Workflow Management System", Proceedings of the EDBT Workshop on Workflow Management Systems, pp. 29-38, Valencia, Spain, March 1998

WEIHMAYER, R., VELTHUIJSEN, H., "Intelligent Agents in Telecommunications", in Jennings, N., Wooldridge, M. (Eds.), "Agent Technology: Foundations, Applications, and Markets", Springer, 1998

WESTON, R., COUTTS, I., "Model Enactment Based on the use of the CIM-BIOSYS Integrating Infrastructure", International Conference on Automation, Robotics, and Computer Vision, Singapore, 1994

WFMC (Workflow Management Coalition), "The Workflow Reference Model", Document Number TC00-1003, http://www.wfmc.org/, 1995

WFMC (Workflow Management Coalition), "Workflow Client Application (Interface 2) Application Programming Interface (WAPI) Specification", Document Number TC-1009, version 1.1, http://www.wfmc.org/, 1996

WFMC (Workflow Management Coalition), "Audit Data Specification", Document Number TC-1015, http://www.wfmc.org/, 1998

WFMC (Workflow Management Coalition), "Workflow Management Application Programming Interface (Interface 2 & 3) Specification", Document Number TC-1009, version 2.0, http://www.wfmc.org/, 1998

WFMC (Workflow Management Coalition), "Interface 1: Process Definition Interchange", Document Number TC-1016-P, http://www.wfmc.org/, 1999

WFMC (Workflow Management Coalition), "Terminology & Glossary", Document Number TC-1011, http://www.wfmc.org/, 1999

WFMC (Workflow Management Coalition), "Interoperability Internet e-mail MIME Binding", Document Number TC-1018, http://www.wfmc.org/, 2000

WFMC (Workflow Management Coalition), "Interoperability Wf-XML Binding", Document Number TC-1023, http://www.wfmc.org/, 2000

WFMC (Workflow Management Coalition), "XML Process Definition Language", Document Number TC-1025, version 0.03 (Draft), http://www.wfmc.org/, 2001

WILLIAMS, T., RATHWELL, G., LI, H. (Eds.), "A Handbook on Master Planning and Implementation for Enterprise Integration Programs", Report Number 160, Purdue Laboratory for Applied Industrial Control, 1996

WILSON, B., "JXTA", New Riders Publishing, 2002

WODTKE, D., "Modellbildung und Architektur von verteilten Workflow-Management-Systemen", PhD Thesis, Universität des Saarlandes, Germany, 1996, published by infix-Verlag, 1997

WODTKE, D., WEIKUM, G., "A Formal Foundation for Distributed Workflow Execution Based on State Charts", Proceedings of the 6th Intl. Conference on Database Theory (ICDT'97), Delphi, Greece, January 1997

WOELK, D., HUHNS, M., TOMLINSON, C., "InfoSleuth Agents: The Next Generation of Active Objects", Object Magazine, July/August 1995

WOOLDRIDGE, M., JENNINGS, N., "Intelligent Agents: Theory and Practice", Knowledge Engineering Review, vol. 10., no. 2., pp. 115-152, 1995

XU, D., WANG, H., "Multi-agent collaboration for B2B workflow monitoring", Knowledge-Based Systems, vol. 15, no. 8, pp. 485-491, November 2002

YANG, J., PAPAZOGLOU, M., "Interoperation Support for Electronic Business", Communications of the ACM, vol. 43, no. 6, pp. 39-47, June 2000

YEE, A., "Order Out of Chaos: Understanding B2B Integration Patterns", http://b2b.ebizq.net/ebiz_integration/yee_1.html, 2000

YONGJIE, R., "An Agent-based Workflow Model", Proceedings of the 4th Doctoral Consortium on Advanced Information Systems Engineering, Barcelona, Spain, June 16-17, 1997

YOO, J.-J., LEE, D., SUH, Y.-H., LEE, D.-I., "Scalable Workflow System Model Based on Mobile Agents", Intelligent Agents: Specification Modeling and Application, Lecture Notes in Artificial Intelligence, vol. 2132, pp. 222-236, Springer, 2001

ZAKON, R., "Hobbes' Internet Timeline v5.3", http://www.zakon.org/robert/internet/timeline/, 2001

ZARLI, A., POYET, P., "A Framework for Distributed Information Management in the Virtual Enterprise: The VEGA Project", pp. 293-306, in [Camarinha-Matos and Afsarmanesh, 1999a]

ZELM, M., "XML Representation of CIMOSA Models", UEML Interest Group of the IFAC-IFIP Task Force, Paris, December 1999

ZHANG, M., LI, W., "Persisting Autonomous Workflow for Mobile Agents Using a Mobile Thread Programming Model", Approaches to Intelligent Agents, Lecture Notes in Artificial Intelligence, vol. 1733, pp. 84-95, Springer, 1999

ZHOU, M., VENKATESH, K., "Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach", Series in Intelligent Control and Intelligent Automation, vol. 6, World Scientific, 1999

ZHUGE, H., CHEN, J., FENG, Y., SHI, X., "A federation-agent-workflow simulation framework for virtual organisation development", Information & Management, vol. 39, no. 4, pp. 325-336, 2002

ZISMAN, M., "Representation, Specification and Automation of Office Procedures", PhD Thesis, Warton School of Business, University of Pennsylvania, 1977

Appendix A

The European Innovation Relay Centre Network

The Innovation Relay Centre (IRC) Network[135] is an initiative supported by the European Commission. The mission of the IRC Network is to stimulate transnational technology transfer (sometimes abbreviated to TTT) between SMEs in Europe. The IRC Network promotes technology transfer in the form of licensing agreements, joint venture agreements, manufacturing agreements, and commercial agreements with technical assistance. The IRC Network comprises about seventy Innovation Relay Centres (IRCs) across 30 European countries. Each IRC provides a range of specialized services to SMEs in its geographical area. Most IRCs are operated by a consortia of qualified regional organizations such as Chambers of Commerce, Regional Development Agencies and Technology Centers. The staff of each IRC includes experienced specialists with backgrounds in business, industry or research.

The range of services provided by an IRC are also available to other organizations such as large companies, research institutes and universities. These services include:

- *Advice on innovation, technology transfer and exploitation* - An IRC can visit an organization and carry out a technology assessment for that organization. Based on this technology assessment, the IRC can advise the enterprise on the opportunity for introducing new technologies or for promoting innovative technologies.

- *Identification of technology needs or technology offers* - IRCs can help enterprises find technology solutions for their business or promote their own innovative products. For this purpose, an IRC can help an enterprise create a technology profile in the form of a *Technology Offer* (TO) or a *Technology Request* (TR). Afterwards, the IRC publishes the technology profile on the IRC Network.

- *Search for European partners* - The technology profiles published by IRCs can be used to search for business partners via the IRC Network, via trade missions, or via technology brokerage events. In the first case, the IRC searches for matching technology profiles published by other IRCs; in the second case, the IRC can arrange meetings with potential business partners; and in the third case, the IRC can promote technologies or even represent an enterprise in exhibitions, trade fairs, and partnering events.

[135] http://irc.cordis.lu/

- *Advice on financial support and intellectual property rights* - IRCs can organize meetings with venture capital funds operators, and can assist enterprises in preparing business plans for investors. In addition, IRCs can provide advice on intellectual property rights; this can be done either by the IRC staff or, if necessary, by a third-party patent lawyer hired by the IRC.

- *Assistance with contract negotiation* - IRCs can help enterprises draft technology transfer agreements, and they can organize the first meeting between two business partners, providing the venue and a translator, if necessary. IRCs can also direct enterprises to specialists who can estimate the market value of a new technology.

The most interesting feature of IRCs is that they are able to (1) help an enterprise create a standard technology profile and to (2) match that technology profile with other technology profiles created by enterprises located across Europe. For this purpose, IRCs rely on a central database which is at the IRC-IRE[136] Central Unit in Luxembourg. This central database is where IRCs publish all technology profiles (TOs and TRs); it is an up-to-date repository of information where IRCs, and IRCs only, can search for matching technology profiles. The structure of the IRC Network is shown in figure A.1: each enterprise must connect to the nearest IRC, which is the interface to the IRC Network; on its turn, each IRC cooperates with other IRCs via the IRC-IRE Central Unit in order to fulfill technology needs and promote technology offers.

IRCs can guarantee that the identity of an enterprise can remain confidential for as long as they desire. Actually, the technology profiles (TOs and TRs) do not reveal the identity of the enterprise; instead, each technology profile just refers to the IRC that published it. Whenever an IRC finds a TO

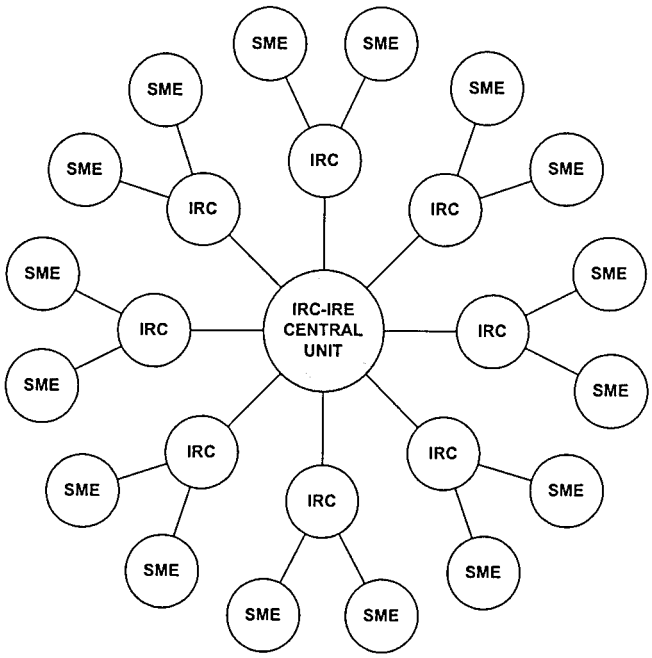


Figure A.1: Structure of the European IRC Network

[136]IRE stands for "Innovating Region in Europe".

that matches one of its own TRs, or a TR that matches one of its own TOs, the IRC forwards that technology profile back to the original enterprise that created the TR or the TO, respectively. If the enterprise expresses interest in the technology profile that has just been found, the IRC gets in contact with the remote IRC that published the technology profile in order to obtain additional information. The remote IRC may or may not reveal the identity of the enterprise it represents, depending on what the enterprise has arranged with the IRC. However, if both enterprises become interested in getting into contact with each other, their IRCs will gladly allow them to do so. In fact, both IRCs will eventually withdraw their role as intermediaries in order to allow both enterprises to do business with each other. The only thing that IRCs require is that enterprises keep them informed of the status and results of their business relationships.

Figure A.2 illustrates an example. Enterprise SME2 has created a TR (step 1), which IRC2 publishes in the IRC-IRE Central Unit database (step 2). At the same time, SME1 creates a TO (step 3), which IRC1 publishes in the central database. When IRC2 checks the database, it finds that there is a TO that matches the TR previously published (step 5). Therefore, IRC2 forwards this TO to SME2 (step 6). If SME2 finds that the TO is interesting and wants to know more about it, IRC2 will get into contact with IRC1 (step 7). Eventually, if both SME1 and SME2 are interested in exploiting this new business opportunity, they can get in contact with each other (step 8). Any of these enterprises can ask its IRC for assistance or even to represent them during contract negotiation.

Technology profiles (TOs and TRs) are human-readable documents that can be prepared with standard word processing applications. Figure A.3 illustrates the structure of such documents:

- The "technology description" section has a title and an abstract which provides a brief description of the technology being offered or sought. The description field describes the technology with more detail; for a TO, it presents the innovative features as well as the advantages of the technology being proposed, whereas for a TR it presents the requirements and possibly some explanation of why the enterprise is looking for such a technology (e.g. to improve an existing product).

- The "technology keywords" section includes a set of standard technology codes, which are taken from a list of predefined values that range virtually every industry sector. The "stage of

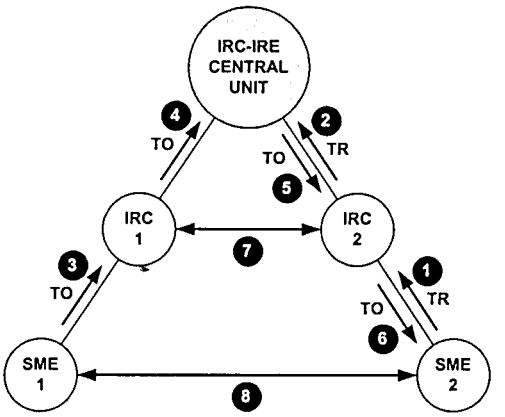


Figure A.2: Matchmaking in the European IRC Network

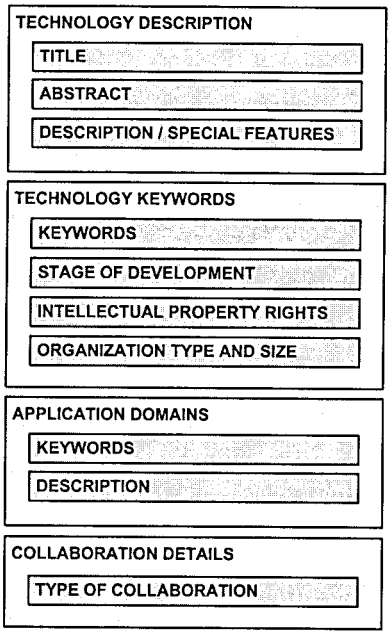


Figure A.3: Structure of technology profiles

development" field is used to determine whether the product is (or should be) under development, if it is (or should be) a prototype, or if it is (or should be) readily available on the market. The "intellectual property rights" field, used only for TOs, is used to specify any patent rights or copyright protection that may apply to the technology being offered. Technology profiles also include the current or desired organization type (industry, technology transfer center, research institute, etc.) and organization size (approximate number of employees).

- The "application domains" section includes standard market application codes for a wide range of sectors such as communications, electronic components, molecular biology, medicine, energy, consumer products, chemicals, transportation and finance.

- The "collaboration details" section specifies the type of collaboration being proposed, such as a joint venture, a commercial agreement, a license agreement, or another option.

The technology profile is the solution that the IRC Network found to the problem of how suppliers and buyers should express their offers and needs, respectively. It is a very interesting solution, and the fact that TOs and TRs are symmetric (they follow the same structure) means that it is possible to search for TOs that match a given TR or to search for TRs that match a given TO. This feature allows IRCs both to find technology offers and to promote them with the same effectiveness. Another important feature is that technology profiles are independent of the structure of the IRC Network itself; an enterprise could hypothetically use technology profiles to describe offers or needs directly to other enterprises. Therefore, TOs and TRs could be used to express technology offers and needs in marketplaces other than the IRC Network. Conversely, the IRC Network could adopt a more decentralized structure while making use of the same kind of technology profiles. As a matter of fact, it has been suggested that peer-to-peer networking could be used to link IRCs directly to each other [Watson, 2001].

BIBLIOTECA F.E.

The Internet allows companies to integrate their business processes with those of its business partners, as well as to discover and establish business relationships with other business partners. This new business environment encourages companies to develop arbitrary business structures, referred to as business networks. As companies evolve into business networks, appropriate IT solutions will be required to address the need to integrate, automate and manage business processes that extend across several enterprises. At the level of application integration, technologies such as message-oriented middleware, business-to-business frameworks, agent-based systems, Web services, or even peer-to-peer networking will allow enterprises to interact with each other. At the level of business process integration, workflow management systems will not only support coordination, but they will also allow enterprises to automate the whole life cycle of their business relationships...

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000079671

ISBN 972-9051-32-1