

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Arquitecturas de Integração de Mediadores com Companhias de Seguros

Miguel Gomes Rodrigues

Relatório de Projecto

Mestrado Integrado em Engenharia Informática e Computação

Orientador: João António Correia Lopes (Professor Auxiliar)

3 de Março de 2009

Arquitecturas de Integração de Mediadores com Companhias de Seguros

Miguel Gomes Rodrigues

Relatório de Projecto

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: Ana Paula Cunha da Rocha (Professora Auxiliar)

Arguente: José Carlos Leite Ramalho (Professor Auxiliar)

Vogal: João António Correia Lopes (Professor Auxiliar)

20 de Março de 2009

Resumo

Há muito tempo que as empresas entenderam a necessidade de informatizar o seu negócio, com o intuito de agilizar os processos a ele inerentes e consequentemente prestar um serviço de maior qualidade. Durante muito tempo, as diversas aplicações de uma determinada empresa funcionaram de forma isolada. Disponibilizar informação para o exterior era impensável. A globalização, as pressões dos parceiros de negócio e a forte expansão do comércio electrónico vieram alterar as mentalidades. Hoje em dia, uma visão integrada do negócio dos parceiros estratégicos é fundamental e inevitável — A integração de software torna-se uma necessidade para os mercados contemporâneos.

A Engenharia de *Software* tem investido imensos esforços nesse sentido. No entanto, actualmente ainda não existe uma metodologia sistemática que possa ser aplicada para solucionar qualquer problema de integração. A integração de *software* é um problema complexo que tem de ser analisado caso a caso.

Solucionar o problema de integração de mediadores com companhias de seguros na empresa I2S é o grande desafio do trabalho reportado nesta dissertação, propondo para esse efeito uma arquitectura de integração. A definição dessa arquitectura envolve uma análise cuidadosa do que pretende ser o processo de integração, estabelecendo os aspectos que devem ser considerados para alcançar um grau de integração satisfatório.

São destacadas duas possíveis arquitecturas como candidatas a solução — Ligações ponto-a-ponto baseadas em *web services* e arquitectura em *bus* providenciada pela tecnologia *Enterprise Service Bus*. De seguida são avaliadas separadamente as duas alternativas e as tecnologias a elas inerentes, através de casos de estudo e provas de conceito. Finalmente, são comparadas as duas soluções, evidenciando as vantagens e pontos fracos de cada uma com base nos aspectos julgados relevantes para a tomada de decisão.

Para concluir, faz-se primeiramente uma síntese do trabalho efectuado ao longo deste projecto, apreciando a satisfação dos objectivos. De seguida, são feitas as considerações finais acerca da integração de *software* e finalmente perspectivam-se os trabalhos que decorrerão futuramente.

Abstract

Long ago, enterprises understood the need to make their business based on information technologies, in order to make more agile their business processes and provide a fully quality service. For long time, enterprise applications ran in an isolated way. Turning information available to the outside was unthinkable. Globalization, business partners' lobbies and the growing up of *e-commerce* call for a change. Nowadays, an integrated vision of partner's business is essential and inevitable — Software integration turn into a need for contemporaneous markets.

Software Engineering has been making many efforts in this way. However, currently, there is no systematic methodology which can be applied to any integration problem. Software integration is a complex problem which need to be analyzed case by case.

Solve the integration problem between brokers and insurance companies at I2S is the main purpose of the work reported in this dissertation, by proposing an integration architecture. The definition of this architecture implies a careful analysis of what integration process should be, establishing the points which should be considered to achieve an acceptable degree of integration. There is two possible architectures investigated as solution — Point-to-point connections based on *web services* and bus architectures provided by *Enterprise Service Bus* technology. The two architectures are evaluated separately from case studies and concept proofs. Finally, they are compared highlighting the strength and weakness of each one based on points considered relevant to the decision to be taken.

A summary of the work done throughout this project is presented in order to verify that the original goals were fulfilled and final considerations are made about the software integration. To conclude, future work is enumerated.

Resumé

Il y a longtemps que les entreprises ont compris le besoin d'informatiser leur métier, dans le but de rendre plus agiles les processus qui en découlent et par conséquent fournir un service de meilleure qualité. Durant longtemps, les diverses applications d'une entreprise donnée, ont fonctionnées de manière isolée. Fournir de l'information propre au métier à l'extérieur était jusque là impensable. La globalisation, les pressions exercées par les partenaires et la forte expansion du commerce électronique ont grandement contribué au changement de la donne. De nos jours, une vision intégrée du métier des partenaires stratégiques est fondamentale et inévitable — L'intégration logicielle devient une nécessité pour les marchés contemporains.

Le Genie Logiciel a investi, en ce sens, de très grands efforts. Néanmoins, actuellement il n'existe pas de méthodologie systématique qui puisse être appliquée à la résolution de n'importe quel problème. L'intégration logicielle résulte en un problème complexe qui doit être analysé cas par cas.

Résoudre un problème d'intégration entre médiateurs et compagnies d'assurances au sein de l'entreprise I2S, est le grand défi du travail rapporté dans cette dissertation, proposant, à cet effet, une architecture d'intégration. La définition de cette architecture englobe une analyse rigoureuse de ce que doit être le processus d'intégration, établissant les aspects qui doivent être tenus en compte pour atteindre un degré d'intégration satisfaisant.

Dans un premier temps, deux architectures envisageables sont mises en évidence — Connexions point à point basées sur les services *web* et une architecture en *bus* soutenue par la technologie *Enterprise Service Bus*. Dans un second temps, les deux alternatives sont évaluées séparément ainsi que les technologies qui les accompagnent, à travers de cas d'étude et de preuves de concept. Finalement, les deux solutions sont comparées, en soulignant les avantages et points faibles de chaque une, en prenant pour base les aspects jugés pertinents à la prise de décision.

En guise de conclusion, une synthèse du travail réalisé au long de ce projet est faite, en appréciant la satisfaction des objectifs. S'en suivent, les considérations finales à propos de l'intégration logicielle et finalement sont établies les perspectives de travaux futurs.

*Aos meus maravilhosos pais, Armindo e Deolinda, e
à minha linda namorada, Susana.*

Agradecimentos

O projecto de conclusão de curso é uma etapa importante na minha vida, uma vez que dita o fim de um longo percurso académico e marca o início da vida profissional. Durante este primeiro contacto com o mundo empresarial tive a sorte de encontrar pessoas que me auxiliaram no dia a dia. A todas as pessoas que de alguma forma deram o seu contributo, queria deixar a minha gratidão.

Em primeiro lugar quero agradecer ao meu orientador, o Doutor João Correia Lopes pela sua orientação esclarecida e rigorosa, pelos seus conselhos pertinentes e pela celeridade com que sempre atendeu as minhas dúvidas. O seu optimismo e a confiança que me testemunhou ajudaram-me a lidar com as situações mais complicadas, principalmente aquando da escrita da dissertação.

Queria também agradecer aos meus colegas de trabalho em geral e em particular ao Mário Rodrigues (Super Mário) pela sua boa vontade e pelos seus preciosos conselhos quando fui levado a trabalhar com a *framework* .NET. Agradeço igualmente ao André Silva pela disponibilidade e paciência que demonstrou ao longo da minha análise da *framework* Mule.

Ao meu amigo Cláudio Sagres pela ajuda proporcionada na revisão deste documento e ao meu amigo Gilles Rodrigues por ter facilitado a minha integração na empresa.

Finalmente, queria deixar um agradecimento especial aos meus pais pelo apoio moral que me deram ao longo deste projecto e por me terem proporcionado a oportunidade, que nunca tiveram, de frequentar o ensino superior.

Miguel Gomes Rodrigues

“Success consists of going from failure to failure wihthout loss of enthusiasm”

Winston Churchill

Conteúdo

1	Introdução	1
1.1	Enquadramento	1
1.2	Projecto	2
1.3	Motivação e Objectivos	3
1.4	Estrutura da Dissertação	4
2	Integração de Software	5
2.1	Visão Global	5
2.1.1	Contexto	5
2.1.2	Definição	6
2.1.3	Problemas e Desafios	7
2.2	Estilos de Integração	7
2.2.1	Invocações remotas	7
2.2.2	Troca de Mensagens	9
2.3	Características e Aspectos a considerar	10
2.3.1	Escolha da tecnologia	10
2.3.2	Manutenção e Reutilização de Código	11
2.3.3	Escalabilidade e Flexibilidade	13
2.3.4	Segurança	14
2.3.5	Sistemas legados	15
2.4	Conclusões	15
3	Web Services — Tecnologia de Referência	17
3.1	Introdução aos <i>Web Services</i>	17
3.1.1	Fracasso das Tecnologias de Distribuição	18
3.1.2	Importância dos <i>standards</i>	18
3.1.3	Definição	18
3.2	Estilos	19
3.2.1	XML-RPC	19
3.2.2	<i>Big Services</i>	20
3.2.3	REST	24
3.2.4	Comparação	25
3.3	Caso de Estudo : VIAPRE	26
3.3.1	Problema	26
3.3.2	Ferramentas e Tecnologias	26
3.3.3	Considerações	28
3.4	Conclusões	28

CONTEÚDO

4	Enterprise Service Bus	30
4.1	Introdução ao <i>Enterprise Service Bus</i>	30
4.1.1	Enquadramento	30
4.1.2	<i>Enterprise Service Bus</i> — Definição	33
4.2	Paradigma: Troca de Mensagens	34
4.2.1	Processamento de Mensagens	35
4.2.2	Filtragem de Mensagens	35
4.2.3	Transformação de Mensagens	36
4.2.4	<i>Routing</i> Inteligente de Mensagens	37
4.3	Caso de Estudo: Provas de Conceito	37
4.3.1	Mule ESB	37
4.3.2	<i>Workflow</i> Básico no <i>Bus</i>	38
4.3.3	Circuito Básico com Segurança Activa	39
4.3.4	<i>web servicesProxy</i>	43
4.4	Conclusões	45
5	Arquitectura de Integração	46
5.1	Justificação do estudo da tecnologia <i>Enterprise Service Bus</i>	46
5.2	Tecnologias Envolvidas	48
5.2.1	Spring <i>Framework</i>	48
5.2.2	Axis	49
5.2.3	CXF	49
5.2.4	WSS4J	50
5.2.5	Acegi	50
5.2.6	JAAS	51
5.2.7	XSLT	51
5.3	Comparação de Arquitecturas	51
5.3.1	Tecnologias	51
5.3.2	Segurança	52
5.3.3	Flexibilidade	52
5.3.4	Escalabilidade	53
5.3.5	Manutenção	54
5.3.6	Sistemas Legados	54
5.3.7	Custo Inicial	56
5.4	Conclusões	56
6	Conclusões	58
6.1	Satisfação dos Objectivos	58
6.2	Considerações Finais	59
6.3	Perspectivas de Trabalhos Futuros	60
	Referências	61

Lista de Figuras

2.1	Invocações Remotas	8
2.2	Troca de mensagens	9
2.3	Ciclo de Vida do <i>software</i>	12
3.1	Arquitectura em Camadas de Serviços SOAP	21
3.2	Serviço de Descoberta de <i>web services</i>	23
4.1	Arquitectura <i>Hub-and-Spoke</i>	32
4.2	Arquitectura em <i>Bus</i>	34
4.3	<i>Workflow</i> Básico dentro do <i>Bus</i>	39
5.1	Arquitectura de Integração — Ligações Ponto-a-Ponto	47
5.2	Exemplo de Invocação Complexa de <i>Web Services</i>	53

Lista de Tabelas

4.1	<i>Standards</i> Usados nos ESB's	33
-----	---	----

Abreviaturas e Símbolos

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
ESB	Enterprise Service Bus
EAI	Enterprise Application Integration
EIP	Enterprise Integration Patterns
FTP	File Transfer Protocol
GIS	Gestão Integrada de Seguros
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
JMS	Java Message Queue
JAAS	Java Authentication and Authorization Security
JRE	Java Runtime Environment
MOM	Message-Oriented Middleware
OASIS	Organization for the Advancement of Structured Information Standards
OO	Object-Oriented
POJO	Plain Old Java Object
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RPG	Report Program Generator
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SMTP	Simple Mail Transfer Protocol
UDDI	Universal Description Discovery and Integration
VS2005	Visual Studio 2005
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformation
W3C	World Wide Web Consortium
WS-Security	Web Services Security
WSDL	Web Services Description Language
WWW	<i>World Wide Web</i>

Capítulo 1

Introdução

Este capítulo introdutório apresenta o enquadramento do projecto, identifica os problemas decorrentes, os objectivos que se pretende atingir e define a estrutura deste documento.

1.1 Enquadramento

Desde sempre, o Homem precisou de controlar o risco. Já no século 23 antes de Cristo, na Babilónia antiga, os camaleiros efectuavam mutualismos entre si para minimizar os prejuízos relativos à morte de animais[[Wik09a](#)]. Nas sociedades actuais, caracterizadas pelo consumismo, a necessidade de controlo do risco é cada vez mais acentuada. O conceito de seguro está omnipresente, torna-se necessário garantir que a perda de bens não causará prejuízos elevados, pelo que, é muito comum subscrever seguros, para a casa, para o carro, entre outros. Hoje em dia todos os grandes mercados estão altamente informatizados, quer para benefícios internos, quer para a prestação de um serviço de alta qualidade. O sector segurador não é excepção, e nele, as tecnologias de informação desempenham mesmo um papel fundamental. Note-se que foram os bancos e as seguradoras os primeiros a informatizar o seu negócio, já na década de 50. As preocupações da época recaíam essencialmente na redução de tempo despendido pelos colaboradores a efectuar tarefas de verificação e cálculo, armazenamento e classificação de informação. Hoje em dia, a realidade das soluções informáticas destas empresas é muito mais complexa do que há 60 anos. As empresas deste ramo, gerem quantidades consideráveis de informação e precisam cada vez mais de disponibilizar serviços a outras entidades envolvidas no seu negócio.

A empresa I2S¹ concebe, desenvolve e implementa software para o mercado global de seguros desde 1984, actuando principalmente no mercado português onde se apresenta como líder mas, marca igualmente presença no Brasil e em África. A I2S propõe soluções de gestão integrada para os diversos intervenientes da actividade seguradora, como companhias, mediadores, peritos ou sociedades de fundos de pensões. Um tomador de seguros é cliente de uma companhia de seguros e poderá eventualmente ser cliente de um mediador de seguros, pelo que se torna fundamental, para estes dois fornecedores de serviços, possuir informação sincronizada, seja ela, inerente ao tomador ou relativa ao seu próprio negócio, o que implica necessariamente a troca dessa informação.

O projecto que dá origem a esta dissertação consiste precisamente na integração de aplicações de software, isto é, pretende-se aumentar o grau de integração da solução das companhias com a solução dos mediadores de seguros. Este documento tem o intuito de descrever como pode ser levada a cabo essa integração, analisando e comparando duas arquitecturas possíveis, de acordo com os meios tecnológicos disponíveis actualmente.

1.2 Projecto

Um pouco antes do ano 2000, a Internet iniciou uma considerável expansão que ainda hoje perdura. Com esse forte crescimento, começou a fazer sentido a troca de informação entre aplicações que não se encontravam na mesma rede local. No sentido de permitir essa interligação, a Engenharia de *Software* empenhou-se em investigar e criar a tecnologia² necessária para montar o cenário há pouco referido. Ao longo da última década, foram aparecendo novas tecnologias de distribuição que permitem a integração de aplicações que inicialmente não foram pensadas para comunicar.

O GIS Mediadores é uma das aplicações de software do pacote GIS, oferecido pela I2S. Esta solução consiste numa aplicação de software em ambiente *Windows* que visa favorecer a gestão da actividade global de mediadores de seguros. A troca e sincronização de informação entre mediadores e companhias de seguros é de elevada importância para o negócio de ambos. É de notar que essa troca de informação já existe, embora seja feita de forma desconectada através do envio periódico de ficheiros, via correio electrónico ou via FTP. Em outras situações, é mesmo necessário efectuar um telefonema ou enviar um fax para se poder confirmar se determinado acontecimento já sucedeu, verificando-se neste caso total ausência de integração e necessidade de intervenção humana.

A aplicação GIS Mediadores, foi desenvolvida na década de 90. Segue uma arquitectura cliente/servidor implementada com base na linguagem C++, sendo a comunicação entre cliente e servidor conseguida com recurso à tecnologia de distribuição DCOM. A

¹Página oficial disponível em : <http://www.i2s.pt>

²Entende-se aqui por tecnologia, tecnologias de distribuição.

troca de informação referida anteriormente é proporcionada pelos módulos de transferência de dados (Pacote Transfer). As principais funcionalidades desse pacote são:

- integração de novas apólices;
- tomadores;
- novas propostas;
- recibos emitidos, cobrados e liquidados.

Actualmente, o único módulo do pacote Transfer que funciona de forma fiável e continuada, é o módulo TransRec que permite o envio de recibos emitidos.

Especialmente nos últimos anos, tem havido um grande empenho das seguradoras na criação de sistemas de acesso interactivo para mediadores (*extranets*). No entanto, estas soluções nem sempre são ideais para as empresas que trabalham com um leque alargado de seguradoras, destacando os seguintes inconvenientes [Afo09].

- os colaboradores são obrigados a dominar vários sistemas diferentes;
- é necessário o conhecimento geral dos códigos de acesso;
- não é possível, geralmente, segmentar o acesso em função do perfil de cada utilizador.

A necessidade de troca de informação, a falta de fiabilidade de certos módulos do pacote Transfer e os inconvenientes dos sistemas *extranet* fornecidos pelas seguradoras conduzem a uma necessidade de integração óbvia. A I2S pretende aumentar o grau de integração da solução GIS Mediadores com as companhias de seguros de modo a que estes possam otimizar os seus processos de negócio, prestando ao mesmo tempo um serviço de maior qualidade aos seus clientes.

1.3 Motivação e Objectivos

Quando se pretende efectuar integração de software, deve-se pensar cuidadosamente a forma como se vai atingir essa integração. Uma boa integração deve resultar em aplicações flexíveis a futuros desenvolvimentos, sem comprometer o bom funcionamento actual.

É certo que actualmente os serviços web, (mais conhecidos como *web services*) são a tecnologia mais usada quando se trata de integração de aplicações dispersas na rede global. Uma primeira abordagem ao problema (a mais óbvia), seria desenvolver do lado das companhias um conjunto de serviços, usando a tecnologia referida anteriormente. Do lado dos mediadores, seria necessário gerar o código de invocação (*proxies*) desses serviços.

Os últimos anos viram o nascimento de outras tecnologias, igualmente com o intuito de resolver problemas de comunicação entre sistemas heterogéneos. Um *Enterprise Service Bus* consiste numa tecnologia de *middleware* cujo conceito se assemelha ao dos *buses* conhecidos na área da Electrónica. O seu objecto é interligar vários componentes, fazendo circular informação entre eles. O *bus* age como um mediador entre fornecedores e consumidores de serviços e tem ainda a capacidade (entre outras) de efectuar *routing* inteligente, assim como transformação de mensagens.

Este projecto consiste na análise e na comparação destas duas arquitecturas como solução para o problema apresentado, com particular ênfase no *Enterprise Service Bus*. Pretende-se analisar detalhadamente as características desta recente tecnologia, evidenciando os seus pontos fortes e deficiências, e procurando os benefícios e desvantagens que advêm da sua utilização, por forma a avaliar a viabilidade da sua utilização na I2S.

1.4 Estrutura da Dissertação

Para além desta introdução, a dissertação contém mais cinco capítulos. O Capítulo 2 refere-se à integração de *software* começando por definir o que se entende por integração, apresentando de seguida as características que devem ser tidas em conta, no âmbito deste projecto, para se alcançar um grau de integração satisfatório. Os capítulos seguintes apresentam e avaliam as duas possíveis arquitecturas. O Capítulo 3 debruça-se sobre as ligações ponto-a-ponto providenciadas pelos *web services*, enquadrando e definindo a tecnologia, evocando os estilos de *web services* que são usados actualmente e exemplificando a tecnologia referida através do relato dum caso de estudo. O Capítulo 4, quanto a si, discute as arquitecturas em *bus*. Primeiramente, faz uma breve introdução desta recente tecnologia, segue-se uma análise ao paradigma central dos *Enterprise Service Bus* — Troca de mensagens e finalmente transcreve as provas de conceitos desenvolvidas ao longo do projecto. Uma vez estas duas arquitecturas analisadas, são em seguida, comparadas no Capítulo 5 que pretende, primeiramente, justificar o estudo da tecnologia *Enterprise Service Bus* e seguidamente, comparar as duas arquitecturas segundo os critérios enunciados no Capítulo 2. Conclui-se este trabalho com o Capítulo 6, onde são relatados, a satisfação dos objectivos, as considerações finais acerca da integração de *software* e a perspectiva de trabalhos futuros.

Capítulo 2

Integração de Software

Este capítulo dedica-se à análise de um problema actual da Engenharia de *Software*: a integração de aplicações informáticas. Num primeiro tempo é dado ao leitor uma visão global do problema. Num segundo tempo, são destacados alguns dos aspectos a considerar antes de se decidir sobre a arquitectura de integração a adoptar. A lista apresentada não é exaustiva, são apenas considerados os aspectos julgados relevantes no âmbito deste projecto.

2.1 Visão Global

Esta secção pretende conferir ao leitor uma visão global do processo de integração. Inicia, contextualizando o aparecimento da integração de aplicações empresariais. Segue-se uma definição do conceito. Por fim é feita uma apresentação dos problemas e desafios encarados por esta área da Engenharia de *Software*.

2.1.1 Contexto

Durante anos, as aplicações empresariais executaram de forma isolada. A informação intrínseca a uma determinada empresa era acedida apenas internamente. Cada empresa dispunha de um conjunto de aplicações para execução de tarefas específicas. As aplicações eram de natureza diversa e apresentavam uma forte heterogeneidade, quer seja em termos arquitecturais (*Standalone vs Cliente/Servidor*) ou em termos tecnológicos (linguagem de programação usada para o desenvolvimento). Eram especializadas e apresentavam geralmente um alto desempenho na execução das suas tarefas (gestão da produção, gestão de salários, vendas etc.) [Pin03]. Em suma, cada empresa era autónoma, possuía um conjunto de aplicações informáticas, que executavam isoladamente sem troca de informação entre elas. Não havia nenhum conhecimento do negócio de outras empresas ou

parceiros estratégicos, era impensável a disponibilização, para o exterior, de informação inerente ao negócio.

Ao longo dos anos, o mundo empresarial sofreu mudanças. Hoje em dia assiste-se, constantemente, a aquisições, fusões de empresas, ou mais simplesmente, parcerias. A partilha e troca de informação, que durante anos se julgou impensável, torna-se, actualmente, necessária e inevitável. A descentralização das empresas e consequente disparidade geográfica, justificaram esforços investidos na tentativa de integração. A necessidade para as empresas, de possuir uma visão integrada do negócio dos parceiros conduziu à tomada de consciência por parte das empresas que a disponibilização de informação, era essencial na busca da prestação de um serviço de maior qualidade e na agilização dos processos de negócio.

2.1.2 Definição

Em termos gerais, define-se a integração de aplicações empresariais como sendo o uso de software para integrar um conjunto de aplicações informáticas heterogéneas. Esta integração, pretende ligar e combinar pessoas, processos, sistemas e tecnologias, de forma a garantir que as pessoas certas e o processo certo tenham acesso à informação e aos recursos certos, no momento certo [Wik09b] [WDB01]. Embora o conceito de integração esteja estreitamente ligado ao conceito de distribuição (da computação), subsistem umas subtilidades que convém referir. Muitas das aplicações empresariais são aplicações *n-tier*, isto é, compostas por várias partes que executam em vários computadores. Numa primeira abordagem superficial, poder-se-ia considerar que estas aplicações estão integradas, o que não é verdade, pelos seguintes motivos [GH05a]:

- As partes comunicantes estão fortemente ligadas (*tightly coupled*), dependem directamente umas das outras;
- Comunicam sempre de forma síncrona.

Pelo contrário, aplicações integradas funcionam de forma independente, uma vez que, estão fracamente ligadas (*loosely coupled*), tais sistemas podem comunicar de forma assíncrona, o que lhes permite delegar tarefas a outras aplicações sem ter de aguardar por uma resposta.

A definição apresentada nesta subsecção, refere-se à integração de processos de negócio de diversas empresas. Este tipo de integração também é referido na literatura informática como, Integração B2B (*Business-to-Business Integration*). Existem outros tipos de integração (A2A ou ASP), que ultrapassam o âmbito deste projecto e não serão discutidos. A origem deste projecto deu-se da necessidade de alcançar um maior grau de integração entre o negócio dos mediadores e companhias de seguros, pelo que a presente dissertação concentra-se na Integração B2B.

2.1.3 Problemas e Desafios

A integração de software é um processo complexo, que implica ponderação sobre um conjunto de aspectos a ter em consideração. Não existe nenhum modelo conceptual de integração, cada negócio é diferente e a integração é um problema que tem de ser atendido caso a caso e para o qual existe quase sempre mais do que uma solução possível e aceitável [GH05b].

Um dos grandes problemas da integração de aplicações empresariais prende-se com o facto de, os vários sistemas a integrar residirem em sistemas operativos diferentes, usarem soluções de bases dados diferentes ou ainda linguagens de programação diferentes. Em certos casos, tem de se contar com os sistemas legados que por vezes já não são suportados por quem os criou inicialmente, o que complica seriamente o processo de integração.

A introdução de conectividade entre aplicações heterogéneas não deve, em nenhum caso, comprometer o funcionamento actual das diversas aplicações. A integração não deve criar dependência entre aplicações nem alterar o desempenho de cada uma delas, quando analisadas isoladamente. Cada sistema, no seio de uma arquitectura de integração, deve poder ser alterado sem que os outros tenham conhecimento, ou sem que a alteração comprometa a comunicação com as outras aplicações [Bus03].

2.2 Estilos de Integração

Não existe uma única forma de alcançar integração de *software*. O leque de tecnologias disponíveis assenta num conjunto de estilos que podem ser adoptados consoante o caso a solucionar. Esta secção, debruça-se sobre dois (invocações remotas, troca de mensagens) dos quatro estilos mais divulgados, considerando que os outros dois (transferência de ficheiros, partilha de base de dados) são irrelevantes para este projecto.

2.2.1 Invocações remotas

A ideia de invocar remotamente procedimentos (RPC) surgiu logo após a criação da Ethernet, em 1975 por Paul Barman. A primeira implementação a ter um verdadeiro sucesso e a ser amplamente utilizada foi a implementação da SUN, actualmente apelidada de ONC RPC, descrita em RFC 1831 e assenta em linguagens procedimentais, como é o caso do C.

O conceito envolvido, ilustrado pela Figura 2.1, prende-se com a chamada de um procedimento distante num processo e espaço de endereçamento diferente, quer seja no mesmo computador quer seja num computador remoto, sem que o programador tenha de se preocupar com os detalhes da interacção propriamente dita. Em suma, a ideia principal é que uma chamada remota seja o mais parecida possível à chamada dum procedimento local [Whi75].

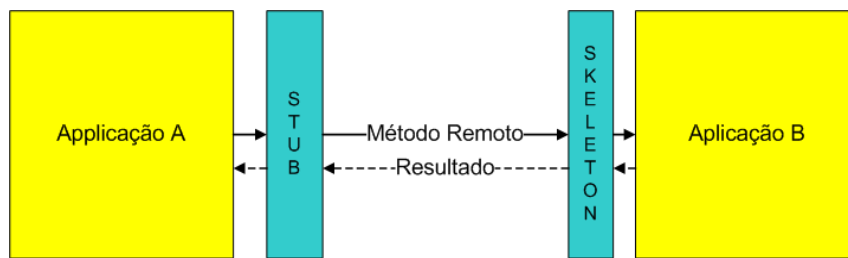


Figura 2.1: Invocações Remotas

A popularidade das linguagens ditas “orientadas a objectos” (OO) contribuiu para o nascimento de outra tecnologia de distribuição; os objectos distribuídos, que não são mais do que invocações remotas para linguagens OO. Os benefícios das linguagens OO para o desenvolvimento de software, conduziram a que as invocações de procedimentos remotos (RPC) se tornassem obsoletas em prol das invocações de métodos remotos (RMI). A ideia básica deste conceito prende-se com a partilha de objectos entre várias aplicações. Um objecto envia uma mensagem a um objecto que se encontra numa máquina remota, pedindo a execução de uma determinada tarefa. Os resultados são devolvidos ao objecto que efectuou o pedido. Do ponto de vista do programador, a execução de métodos sobre objectos remotos é feita de forma transparente, como se de uma invocação local se tratasse.

As tecnologias que respondem a este paradigma e que mais sucesso alcançaram foram; Java RMI, CORBA e COM/DCOM. Criadas inicialmente para serem usadas em ambientes controlados, como redes locais (intranet), estas tecnologias apresentam limitações nomeadamente quando se abordam questões de portabilidade e interoperabilidade. Tentou-se criar um protocolo, Inter-ORB que proporcionasse o uso de tais tecnologias, mais precisamente o CORBA na Internet. Este protocolo nunca conseguiu entusiasmar os programadores que viam nele um protocolo complexo e ainda com muitas deficiências. Estas carências, podem comprometer o processo de integração quando o domínio de integração é um sistema complexo como a Internet.

Os *web services* são hoje uma tecnologia de referência quando se aborda o tema das invocações de métodos remotos. Uma das particularidades que lhes confere um valor acrescido, é o uso de standards como o XML e SOAP e a facilidade com que se integra com o transporte HTTP, que desta forma, favorece o contorno de firewalls [GH05c]. No Capítulo 3 é feita uma análise mais exaustiva a esta tecnologia.

A integração baseada em invocações remotas é facilmente implementada quando a integração apenas envolve duas aplicações e ligações ponto-a-ponto. Num cenário em que existem mais do que duas aplicações de *software* a integrar, o uso de invocações remotas pode comprometer a integração. Na medida é que estas invocações são *tightly coupled*, uma falha de comunicação de uma invocação conduz à falha de toda a cadeia de

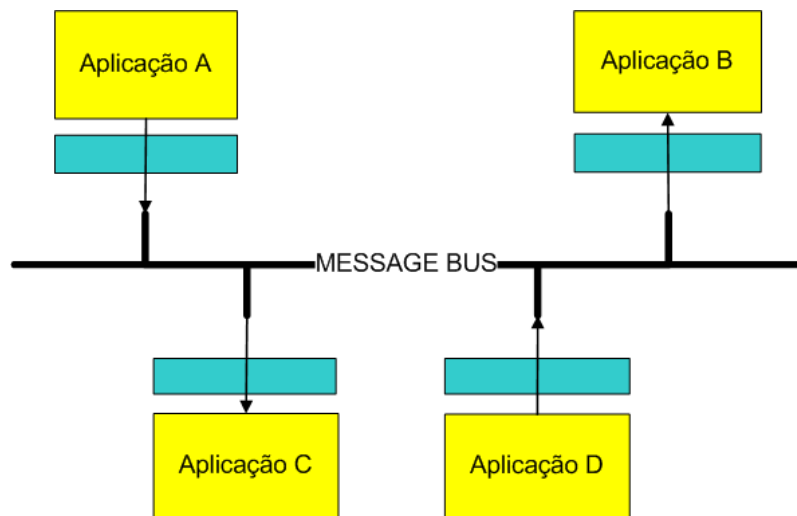


Figura 2.2: Troca de mensagens

invocações subsequentes. Uma das vantagens largamente divulgada, logo que surgiram estas tecnologias era a de execução de um método remoto como se de um método local se tratasse. Na prática, esta afirmação tem de ser revista, considerando apenas o facto de que uma invocação remota tem um tempo de execução muito mais elevado do que uma invocação local.

2.2.2 Troca de Mensagens

A necessidade de remoção, ou pelo menos, diminuição do risco de falhas que advêm dos problemas enunciados na secção anterior (Invocações remotas), deu nascimento a um novo estilo de integração — Integração baseada em troca de mensagens [GH05d].

Este tipo de integração assenta no paradigma de *Message-Oriented Middleware* (MOM), que consiste no uso *software* que reside nos dois lados de uma arquitectura cliente/servidor e suporta chamadas assíncronas entre o cliente e o servidor [Wik09e]. Considerou-se importante desacoplar as diferentes aplicações envolvidas no processo de integração, problema bem patente com as invocações remotas.

O negócio tem as suas especificidades, não existem dois negócios exactamente idênticos. Essa diferenciação é transportada para as aplicações informáticas que dão suporte a esse negócio, de facto, cada aplicação possui uma semântica própria. Nos primeiros tempos de vida do conceito de integração de aplicações empresariais, em que a integração era totalmente “caseira”, era necessário extrair a informação de que se precisava do formato recebido e de seguida re-encapsular essa informação num formato que pudesse ser digerido pela aplicação. Quando uma aplicação comunicava com vários *endpoints*, de

aplicações ou instituições diferentes, tinham necessariamente de ser conhecidos os vários formatos de mensagens de cada aplicação ou empresa.

Este tipo de tecnologia pretende igualmente atender este problema de formatos. É possível efectuar transformação de mensagens antes de estas serem entregues à aplicação de destino. Contudo, mantém-se outros problemas inerentes a este tipo de arquitectura, tais como :

- Como é possível determinar a aplicação de destino?
- Qual o tipo de transporte a usar para entregar a mensagem?
- Que tipo de transformação aplicar à mensagem para que esta seja entendida pela aplicação cliente?

Todas estas questões, ficam por enquanto pendentes, esta secção tem apenas o intuito de dar uma visão geral dos estilos de integração actuais. Serão atendidas mais atentamente no Capítulo 4, uma vez que o tipo de *software* nele discutido usa intensivamente este estilo de integração.

2.3 Características e Aspectos a considerar

Esta secção apresenta as características (aspectos) que devem ser considerados, para se atingir um grau de integração satisfatório.

2.3.1 Escolha da tecnologia

Quando as empresas entenderam a importância de integrar as suas aplicações com as de outras instituições ou parceiros, tiveram de desenvolver a sua própria plataforma de suporte à integração. De facto, a ausência de produtos (*software*) comerciais, testemunhava a falta de interesse dos fornecedores de *software* por esta área, considerando ainda que se tratava de um mercado pouco prolífico.

A portabilidade e a interoperabilidade são conceitos fundamentais quando se aborda este tipo de problema. O primeiro diz respeito à independência da tecnologia em relação à plataforma, geralmente ao sistema operativo. Refere-se a título de exemplo a linguagem *Java*, que executa na maioria dos sistemas operativos em uso actualmente. A interoperabilidade refere-se, à independência de uma tecnologia em relação a outras tecnologias. A tecnologia de distribuição da *Java RMI*, apenas permite a interacção com aplicações *Java*.

Como já foi antes referido neste documento, a integração de *software* é um dos problemas que mais interesse desperta na Engenharia de *Software*. Sensivelmente a partir do ano 2000, com a forte expansão do comércio electrónico, foram produzidos grandes

esforços no sentido de disponibilizar a tecnologia necessária à implementação da integração. As empresas escolhem as tecnologias consoante a adequação destas ao seu negócio. Os critérios de escolha poderão ser:

- *know-how* da empresa com determinada tecnologia;
- carácter livre ou proprietário da tecnologia;
- facilidade de integração com outras tecnologias usadas na empresa.

A escolha da tecnologia não se limita apenas à escolha de uma linguagem de programação. Deve também considerar-se o uso de *frameworks*, que são componentes de *software* suficientemente genéricos para poderem ser reutilizáveis e consequentemente evitar aos programadores a reescrita dessas funcionalidades.

Não é possível para uma empresa prever se terá de integrar o seu negócio com outra empresa, pelo que, à partida não sabe qual vai ser a tecnologia usada por um eventual parceiro, no entanto, uma má escolha da tecnologia poderá comprometer a integração.

Para solucionar o problema de incompatibilidade, entre outros, julgou-se essencial a definição de *standards* para evitar precisamente a impossibilidade de comunicação ou para aliviar processamentos complexos, nomeadamente no que diz respeito a extracção de mensagens usadas por cada uma das aplicações envolvidas na integração, no caso de uma integração baseada na troca de mensagens.

2.3.2 Manutenção e Reutilização de Código

A manutenção de *software* consiste no processo de alteração após este estar em fase de produção. O processo de desenvolvimento de *software* é decomposto em várias fases:

- definição de Requisitos;
- desenho;
- implementação;
- testes;
- manutenção.

A manutenção é a última fase do ciclo de vida de uma aplicação, mas frequentemente exige que se volte à fase inicial, consoante a complexidade das alterações necessárias, como se pode ver na Figura 2.3:

A manutenção de *software* pode também ela ser decomposta em várias fases:

Manutenção correctiva : modificação do *software* em fase de pós-produção de modo a corrigir defeitos detectados.

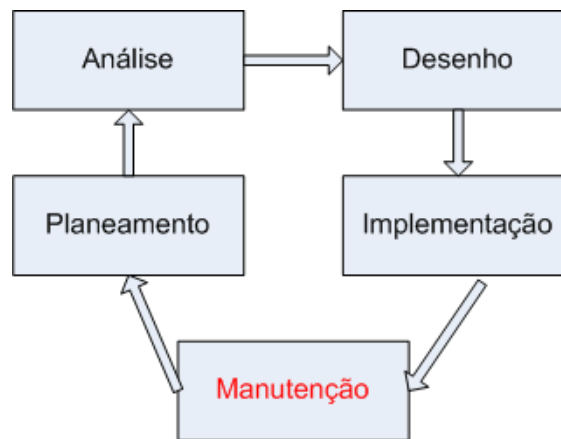


Figura 2.3: Ciclo de Vida do *software*

Manutenção adaptativa : modificação do *software* em fase de pós-produção de modo a manter-se utilizável num ambiente alterado.

Manutenção perfectiva : modificação do *software* em fase de pós-produção de modo a melhorar a eficácia ou manutenção.

Manutenção preventiva : modificação do *software* em fase de pós-produção de modo a detectar e corrigir falhas latentes que ainda não se manifestaram.

A escolha da tecnologia, referida na subsecção 2.3.1, tem uma importância significativa num processo de integração e encontra-se estreitamente relacionada com a manutenção de *software*.

Num cenário em que uma empresa decide não recorrer a *frameworks* optando por uma integração “caseira”, terá uma maior quantidade de código a manter, o que se pode tornar custoso em termos económicos, de tempo e de pessoal. A manutenção de *software* (englobando as 4 fases enunciadas) pode representar até 60% do esforço total para uma empresa de desenvolvimento de *software*, sendo que a correcção de erros representa cerca de 20% [Pre97]. A responsabilidade de manutenção, não tem só aspectos negativos, ao contrário do que leva a crer a última afirmação. Quando é necessária uma correcção, ter-se-à o código disponível para se proceder à respectiva correcção e uma liberdade total para se alterar o código necessário.

No cenário contrário, uma empresa pode optar pelo uso de *software* externo como *frameworks*, para aliviar nomeadamente o tempo de desenvolvimento inicial, apostando na reutilização de código, característica das linguagens *OO* e das *frameworks*. Em primeira instância, esta solução é aconselhável, uma vez que diminui a quantidade de código a manter e consequentemente diminui custos financeiros, de tempo e de pessoal, mas levanta outras questões:

- O que acontece, se após alguns meses é detectada uma falha no *software* de terceiras partes (*third parties*)?
- Qual é o grau de confiança que se pode depositar em *software* de terceiros?

Quando o *software* não é aberto, resta comunicar ao fornecedor e aguardar que seja disponibilizado um correctivo. No caso de se tratar de uma *framework* de código aberto, tem-se sempre a possibilidade de alterar o código (quando existem licenças que permitem essas alterações). No entanto, algures no futuro, a empresa responsável atenderá esses problemas e disponibilizará certamente actualizações e correctivos, o que poderá eventualmente conduzir a conflitos com a distribuição alterada. Deve-se frisar, no entanto, que geralmente este tipo de *software* foi previamente testado e validado pelo que se poderá atribuir um grau de confiança muito aceitável.

2.3.3 Escalabilidade e Flexibilidade

Devido às constantes pressões externas sofridas pelas empresas contemporâneas, às súbitas e inesperadas mudanças no mundo empresarial, torna-se extremamente difícil (impossível?) prever futuras necessidades de integração. Aquando de uma semelhante necessidade, não é aconselhável apenas preocupar-se em solucionar o problema presente mas precaver-se o mais cedo possível para os futuros problemas. Algures no futuro será sem dúvida necessário adicionar parceiros ao negócio, o que terá como consequência o acréscimo de aplicações à arquitectura de integração. A capacidade de uma arquitectura de integração em aceitar a adição de novas aplicações, sem originar alterações significativas à arquitectura actual, é uma característica essencial, à qual se dá o nome de escalabilidade.

A facilidade de abertura de uma aplicação informática a alterações, evidência a sua flexibilidade, e é igualmente uma característica desejável. Neste documento, tem-se referido em várias secções, as mudanças mais ou menos repentinas e mais ou menos drásticas nos mercados globais actuais. A celeridade da resposta com a qual uma empresa responde a estas mudanças, determina uma vantagem competitiva que se pode vir a adquirir em relação aos competidores mais directos. Como consequência, afirma-se que a arquitectura de integração deverá possuir a flexibilidade suficiente para poder atender às alterações necessárias, num prazo de tempo aceitável.

Existem vários aspectos de uma aplicação informática, que se qualificam de configuráveis e que podem ser expressos declarativamente ou contrário do que aconteceu durante muito tempo em que tinham de ser expressos imperativamente, isto é, codificados em duro (*hard-coded*) na aplicação. Na maioria dos casos, recorre-se ao *standard XML* para externalizar os aspectos configuráveis de uma aplicação.

Como ilustração, dá-se o exemplo das ligações a base de dados, como programação imperativa, as ligações estão codificadas em duro no código fonte da aplicação, pelo que a

alteração da base de dados implica alterar todas as referências a essa base no código fonte, e posteriormente a re-compilação da aplicação. Com o uso de programação declarativa, esse problema não se coloca, uma vez que se declara num ficheiro XML as características da base de dados e o código-fonte faz referência ao dito ficheiro. Qualquer alteração terá apenas de ser efectuada no ficheiro de configuração.

Esta abordagem, tem vindo a ser cada vez mais usada, já que reduz o tempo de manutenção de código e aumenta a flexibilidade do *software*. Este tipo de programação deve claramente ser encarado quando se discute integração de *software*.

2.3.4 Segurança

As empresas precisam de garantir que a informação que expõem não será consumida por entidades mal intencionadas. Com *webização*¹ das aplicações informáticas, que ainda está em fase de crescimento, aumentaram as preocupações em matéria de segurança de aplicações informáticas.

Estas questões adquirem proporções ainda maiores quando se trata de integração de *software*. As empresas modernas têm a consciência de que a partilha de informação para o exterior é necessária e inevitável, sob pena de se perder competitividade sobre os concorrentes mais directos. Embora a ideia de expor informação para o exterior seja completamente aceite, levanta um problema de segurança.

Existem vários conceitos relacionados com a segurança das aplicações informáticas, que de seguida se expõem:

- Autenticação;
- Autorização;
- Integridade;
- Confidencialidade.

Fundamentalmente, existem duas formas de se alcançar segurança, aquando da transacção de informação sensível. É possível obter-se segurança a nível da própria mensagem que circula através da rede, usando para este efeito, mecanismos de encriptação. A outra forma de se conseguir uma passagem de dados seguros, passa por garantir a segurança do canal de comunicação. No domínio do *e-banking*, crítico em matéria de segurança, todas as transacções são efectuadas sob o transporte *HTTPS*².

Nos capítulos seguintes serão descritos com mais pormenor vários *standards* e mecanismos de segurança, desenvolvidos ao longo do tempo, quer para o caso da integração baseada em invocações remotas (*web services*), quer para o caso da integração baseada em troca de mensagens.

¹Tendência em substituir as aplicações tradicionais por aplicações *web*.

²Versão segura do protocolo HTTP.

2.3.5 Sistemas legados

Na área das tecnologias de informação, a tendência tem sido, olhar para o futuro e pôr constantemente em causa as limitações e deficiências do que fora desenvolvido no passado [McG05]. O desenvolvimento de *software* tem custos financeiros elevados para uma empresa, principalmente quando esse *software* suporta o negócio da organização, pelo que se exige de uma aplicação robustez suficiente para que se mantenha em funcionamento durante um período de tempo aceitável [Som00]. Note-se que na área de seguros e bancária, os negócios destas empresas assentam em sistemas com várias décadas de existência. Trata-se de sistemas que executam em ambiente *AS/400*. À hora actual, a falta de pessoas com sólidos conhecimentos em linguagem RPG é evidente, assim como a complexidade de tais aplicações e o impacto financeiro, o que dificulta a migração do *software* para outras tecnologias e plataformas mais recentes.

Considerando que tais aplicações conseguiram manter-se em funcionamento em mercados críticos como a banca, a sua robustez é inquestionável. A quantidade e o valor da informação abarcada pelos sistemas legados é considerável e constituem um ponto de renitência quando se aborda a questão da evolução ou rejuvenescimento dessas aplicações. Por outro lado, a integração e manutenção de tais sistemas poderá ser extremamente custosa [Zou02].

Os sistemas legados constituem um dilema no campo da Engenharia de *Software*, a sua manutenção ou a sua evolução deve ser cuidadosamente avaliada, sob risco de incorrer em prejuízos ainda maiores. Em nenhum caso, estes sistemas devem ser esquecidos, sendo que se se justificar mantê-los, o processo de integração deverá ter em consideração estes sistemas e avaliar a melhor forma de conseguir integrá-los na arquitectura.

2.4 Conclusões

Este capítulo teve o intuito de analisar com algum detalhe o processo de integração de *software*, começando por contextualizar o ambiente em que se inserem as empresas para justificar a necessidade de integração e de seguida definindo a integração como sendo, a interacção entre aplicações informáticas heterogéneas.

Actualmente, a tecnologia providenciada pela Engenharia de *Software* permite definir estilos de integração, dos quais foram destacados os dois relevantes para este projecto — Invocações remotas, analisadas em detalhe no Capítulo 3 e troca de mensagens ilustradas no Capítulo 4.

Rapidamente se concluiu que a integração de *software* é um processo complicado que tem de ser cuidadosamente analisado, caso a caso. Para esse efeito, foram definidos os aspectos a ter em conta para este projecto, por forma a se alcançar um grau de integração satisfatório. A escolha da tecnologia utilizada é essencial e tem implicações directas na

manutenção de código, na flexibilidade e escalabilidade da arquitectura de integração e no suporte proporcionado aos sistemas legados. Todos esses aspectos são também eles discutidos e apresentados como relevantes para este projecto. Será em função da satisfação desses aspectos que se elegerá no Capítulo 5 a solução mais adequada ao problema de integração de mediadores com companhias de seguros dentro da empresa I2S.

Capítulo 3

Web Services — Tecnologia de Referência

Os *web services*, são actualmente uma tecnologia de referência quando se aborda questões relacionadas com integração de aplicações de *software*. A portabilidade e interoperabilidade que apresentam contribuem fortemente para o seu sucesso. Este capítulo, proporciona uma apresentação da tecnologia e o seu papel na integração de *software*. Num primeiro tempo, é feita uma introdução aos *web services*, de seguida apresentam-se os diversos estilos que existem actualmente e finalmente, exemplificam-se as características dos serviços SOAP, com o relato de um caso de estudo que consistiu no desenvolvimento de um *web service* para a empresa de peritagens VIAPRE.

3.1 Introdução aos *Web Services*

Dum ponto de vista conceptual, os *web services* não apresentam novidades — pretendem aceder às funcionalidades, remotamente, à semelhança do que acontecia com as tecnologias de distribuição. À hora actual as tecnologias de distribuição caíram em desuso, tornando-se obsoletas, em prol dos *web services*. Nesta secção, é evidenciado em que falharam as tecnologias de distribuição, mais precisamente, quando a rede de comunicação é a *Internet*, qual a missão a que se propõem os *web services*¹ e finalmente justifica-se o sucesso desta tecnologia, com a introdução de *standards*.

¹Note-se que quando se falar de *web services*, se não for expressamente referido, deve entender-se *web services* SOAP.

3.1.1 Fracasso das Tecnologias de Distribuição

Ainda há poucas décadas, adivinhava-se um futuro promissor para as tecnologias de distribuição, nomeadamente DCOM e CORBA. A verdade é que, hoje em dia, estas tecnologias caíram em desuso. A responsabilidade do fracasso pode ser atribuída aos implementadores das duas especificações e as suas promessas ilusórias, em termos de interoperabilidade e portabilidade. Por um lado, o Open Management Group estipulava a forma como deveria ser alcançada a interoperabilidade do CORBA, por outro, os implementadores da especificação estavam preocupados em vender a sua solução nas duas extremidades da computação distribuída — cliente e servidor.

No caso do DCOM, todos os nós da aplicação distribuída têm de executar em ambiente Windows. No caso do CORBA, todos os nós têm de executar o mesmo ORB (*Object Request Broker*). Ambos os protocolos são aceitáveis em ambientes locais, enquanto que, a probabilidade de pôr estes protocolos a funcionar entre duas máquinas dispersas na *Internet* é muito reduzida [Gis01]. Ambos os protocolos, sofrem de problemas de segurança, note-se que, no caso do DCOM o porto de escuta era estático, o que levantava problemas para certas empresas. A extensa especificação que os acompanha torna a distribuição (*deployment*) de aplicações bastante complexo.

3.1.2 Importância dos *standards*

Devido às necessidades de integração das empresas, tornou-se necessário colmatar as deficiências das tecnologias de distribuição. A necessidade de *standards* era óbvia para responder aos problemas enfrentados pelo DCOM ou CORBA. Pretendia-se tomar por base os *standards* já existentes, nomeadamente HTTP e XML e desenvolver outros, como é o caso do SOAP e WSDL, de modo a garantir uma interoperabilidade total. Estes dois últimos são descritos com mais detalhe na secção seguinte.

3.1.3 Definição

Um *web service* é uma aplicação de software desenhada para suportar interacções máquina-máquina interoperáveis, através de uma rede [Boo04], isto é, uma aplicação que permite a troca de informação entre aplicações ou sistemas heterogéneos, em ambientes distribuídos. Outro ponto de vista, consiste em afirmar que, se uma aplicação é acessível remotamente usando para tal uma combinação de protocolos, como HTTP e XML, então essa aplicação é um *web service* [Sne01]. Actualmente, existem dois tipos de *web services* — SOAP Services e RESTful Services, ambos satisfazem a definição do W3C apresentada, e serão apresentados em maior detalhe na Secção 3.2. O principal objectivo desta tecnologia é expor funcionalidade na *Internet* ou *Intranet*, de modo a que possa ser acedida por outras aplicações, sem requerer intervenção humana.

3.2 Estilos

Embora neste projecto, sejam usados *web services* SOAP, uma vez que são actualmente os mais populares e os únicos usados na I2S, por razões de familiaridade e de suporte nos IDE's. Convém, no entanto referir que existem mais dois estilos de *web services* que são de seguida, sumariamente apresentados.

3.2.1 XML-RPC

XML-RPC é um protocolo muito simples (a especificação consiste em duas páginas de texto). Criado em 1998, por Dave Winer, segue o paradigma de invocações remotas. Perante a necessidade de *standards* evocado na introdução deste capítulo, esta tecnologia faz uso de, apenas dois, para garantir interoperabilidade total na comunicação entre duas máquinas através da rede global.

A mensagem que circula na rede é descrita em XML (existe uma variante que faz uso da linguagem JSON), encapsula o nome do método a invocar assim como os parâmetros de entrada. A informação necessária (a que circula) à execução do método adopta o nome de *payload*. O transporte usado, é o HTTP e o protocolo suporta, apenas, um conjunto restrito de tipos de dados:

- array
- base64
- date/time
- boolean
- double
- integer
- string
- struct
- nil

A grande vantagem deste tipo de serviço é a sua simplicidade; veja-se de seguida os exemplos de invocação, de resposta e de falha [Wik09d].

```

1 <?xml version="1.0" ?>
2 <methodCall>
3   <methodName>examples.getStateName</methodName>
4   <params>
5     <param>
```

```

6      <value><i4>40</i4></value>
7    </param>
8  </params>
9 </methodCall>

```

Listing 3.1: XML-RPC — Exemplo de Invocação

```

1 <?xml version="1.0"?>
2 <methodResponse>
3   <params>
4     <param>
5       <value><string>South Dakota</string></value>
6     </param>
7   </params>
8 </methodResponse>

```

Listing 3.2: XML-RPC — Exemplo de Resposta

```

1 <?xml version="1.0"?>
2 <methodResponse>
3   <fault>
4     <value>
5       <struct>
6         <member>
7           <name>faultCode</name>
8           <value><int>4</int></value>
9         </member>
10        <member>
11          <name>faultString</name>
12          <value><string>Too many parameters.</string></value>
13        </member>
14      </struct>
15    </value>
16  </fault>
17 </methodResponse>

```

Listing 3.3: XML-RPC — Exemplo de Falha

3.2.2 Big Services

Os SOAP Services, também apelidados de *Big Services*, são uma evolução lógica dos XML-RPC. São de seguida apresentados os conceitos fundamentais inerentes a estes serviços. A arquitectura destes serviços está organizada em cinco camadas, como ilustrado na Figura 3.1.

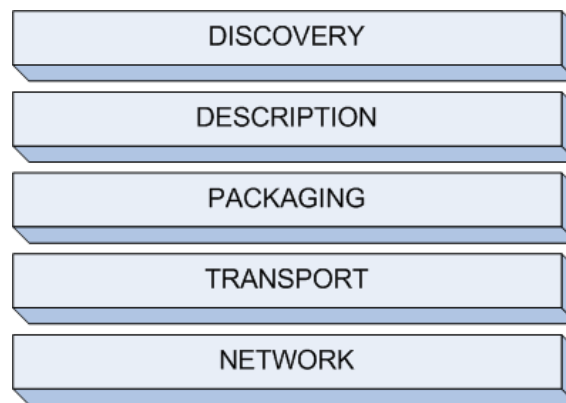


Figura 3.1: Arquitectura em Camadas de Serviços SOAP

3.2.2.1 SOAP

SOAP é um dialecto XML, relativamente recente. Criado em 1998 na versão 0.9. Em Abril de 2000 o protocolo é submetido ao W3C, já na sua versão 1.1. A partir de Setembro de 2000, o protocolo sofreu algumas alterações para dar origem à versão 1.2, ainda utilizada actualmente. Mais de 40 empresas, nas quais se incluem Microsoft, IBM, Sun, Intel participaram activamente na especificação do protocolo, o que demonstra o interesse destes actores no desenvolvimento de *standards* para a comunicações entre aplicações na *Internet*. Por sua vez, o interesse no desenvolvimento de *standards* por parte destes gigantes da indústria do *software*, denota uma alteração de mentalidade, uma vez que estão dispostos a proporcionar a interacção dos seus produtos com os de outros fornecedores, o que tinha sido até aqui impensável, já que a única preocupação, era vender a sua solução. Estes factores, reforçam a ideia que a integração² de negócios se tornava inevitável e inadiável.

O objectivo deste protocolo é a troca de informação estruturada e uniformizada, na medida em que todas as mensagens têm o mesmo formato, numa rede de comunicação de dados, como por exemplo a *Internet*. Na arquitectura apresentada anteriormente, o protocolo SOAP insere-se na camada de encapsulamento (*packaging*).

De seguida apresenta-se uma excerto de mensagem SOAP :

```

1 <?xml version="1.0" ?>
2 <soap:Envelope
3   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5   <soap:Header>
6     ...
7     ...
8   </soap:Header>
9   <soap:Body>
```

²Do ponto de vista informático.

```

10      ...
11      ...
12      <soap:Fault>
13          ...
14          ...
15      </ soap:Fault>
16  </ soap:Body>
17 </ soap:Envelope>

```

Listing 3.4: XML-RPC — Exemplo de Falha

Facilmente se verifica que este excerto está constituído por quatro blocos de informação distinta [W3S]:

- um elemento *Envelope* que identifica o documento XML como sendo uma mensagem SOAP.
- um elemento *Header* que contém informação de cabeçalho (linha 2).
- um elemento *Body* que contém a informação relativa à invocação ou à resposta devolvida (linha 4).
- um elemento *Fault* com informação relativa a erros e falhas (linha 12).

3.2.2.2 WSDL

Uma vez que o serviço está exposto, os eventuais consumidores têm de saber como invocar este serviço, mais precisamente, qual a informação a enviar, qual a informação que será retornada e onde encontrar o serviço. Assim sendo, é então necessária uma interface de acesso para o serviço. A descrição de *web services* insere-se na camada *description* da arquitectura em camadas com ilustrado pela Figura 3.1. Essa especificação fica descrita num ficheiro com extensão WSDL. Este ficheiro contém informação relativa a quatro aspectos distintos dos *web services*: dados (tipos), mensagens, interface, e serviços.

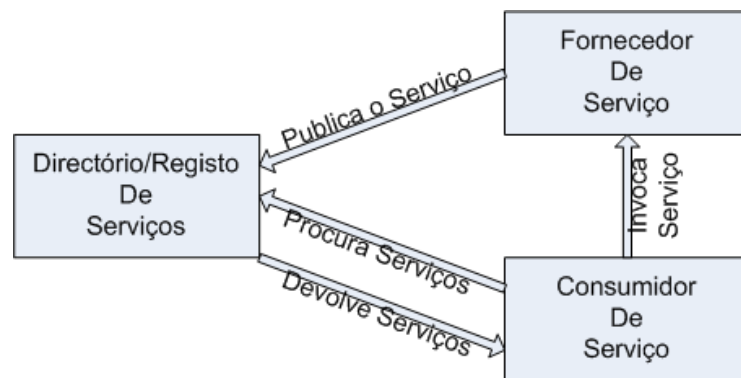
WSDL é também um dialecto XML que permite descrever o *web service* exposto, em termos de :

portType Operação a executar pelo serviço.

message Mensagens usadas pelo *web service*.

types Tipos de dados usados pelo *web service*.

binding Protocolos de comunicação usados pelo *web service*.

Figura 3.2: Serviço de Descoberta de *web services*

3.2.2.3 UDDI e Ligações Ponto-a-Ponto

A camada de topo da arquitectura dos *web services* diz respeito à visibilidade (camada *Discovery* da Figura 3.1). Um serviço deve poder ser descoberto (a pedido de um consumidor), para tal deve ser registado num serviço de directório, de modo a que se possa guardar informação acerca do *web service*.

Aquando da requisição de um *web service*, é estabelecida uma ligação ponto-a-ponto entre a aplicação que fornece o serviço e a aplicação que o consome. Existem dois tipos de ligações ponto-a-ponto que se apresentam de seguida:

ligações estáticas em que o cliente sabe no momento da compilação do código de invocação do serviço (*proxy*) onde se encontra o serviço e os métodos que irá invocar, e assim a localização do servidor pode estar codificada em duro (*hard-coded*). Uma alteração do serviço, implica necessariamente a re-geração de código de invocação, quando se trata do cliente e re-geração da interface (WSDL), do lado do servidor.

ligações dinâmicas em o cliente apenas sabe a localização do registo onde estão localizados os serviços e os métodos que irá invocar no momento da execução. Um consumidor requisita um determinado serviço junto de um directório de serviços (UDDI), que encontra um serviço e devolve a localização ao consumidor, que de seguida trata de ligar com o serviço (*binding*). A este tipo de integração, dá-se o nome de *Just-In-Time Integration*. Uma deslocação do serviço não implica qualquer alteração no código do cliente ao contrário do que acontece no cenário anterior. A Figura 3.2 ilustra o cenário referido.

As ligações ponto-a-ponto adequam-se perfeitamente a projectos de pequena envergadura, apresentando as seguintes vantagens:

eficiência Tem-se uma ligação de elevado desempenho, atendendo ao facto da ligação ser feita directamente entre as partes envolvidas. A troca de mensagens é feita em paralelo.

facilidade de implementação A comunicação bi-direcional é fácil de implementar, uma vez que não existem intermediários.

A desvantagem óbvia, com este tipo de arquitectura é a complexidade de administração. Quando a integração envolve apenas duas aplicações, a complexidade de administração não se coloca, mas quando existem n sistemas, que têm de ser todos integrados ponto-a-ponto o número de ligações é dado por: $n \times \frac{(n-1)}{2}$ [Wik09b] .

3.2.2.4 Transporte

Foi apresentada a linguagem SOAP como sendo a representação universal da informação na utilização de *web services*. Resta então determinar como é encaminhada a mensagem SOAP, entre a máquina de origem e a máquina de destino.

O SOAP não está ligado a nenhum transporte em particular, pode circular em cima de HTTP, SMTP, JMS ou ainda Jabber. No entanto, o mais usado actualmente é o HTTP. A principal razão do sucesso do HTTP, prende-se com a porta de escuta (80). Actualmente, a grande maioria das empresas usa *proxies servers* para filtrar o tráfego de entrada e saída nas fronteiras da sua empresa, sendo que a porta 80 encontra-se sempre aberta, sendo consequentemente a solução ideal para a troca de informação, via *web services*. No entanto, a porta de escuta não é estática, como acontecia com a tecnologia DCOM, é possível usar qualquer porta, desde que a empresa, deseje manter essa porta aberta a ligações vindas do exterior.

3.2.3 REST

REST é o acrónimo de *Representational State Transfer* e consiste num conjunto de regras de utilização da Web, desenvolvidas por Roy Fielding na sua tese de doutoramento [Fie00]. A Web, tal como é conhecida actualmente, não é mais do que um conjunto de recursos (representados sob diversas formas: HTML, XML, PDF, JPEG etc. . .) que podem ser acedidos a partir de aplicações cliente, como por exemplo os navegadores (*browsers*). Cada pedido efectuado pelo cliente coloca-o num determinado estado, consoante o recurso devolvido como resposta.

A implementação de um *web services* segundo a arquitectura REST, implica uma visão da aplicação orientada aos recursos e a classificação da mesma num conjunto de recursos, o que torna a aplicação num depósito de recursos que podem ser requeridos [Lar07].

Um recurso é localizado através do seu identificador único (*Unique Ressource Identifier*), esses URI's são os nomes dos serviços. Existem 4 operações fornecidas pelo protocolo HTTP que são suficientes para interagir com os recursos:

GET Obter a representação de um determinado recurso

PUT Criar um novo recurso.

POST Actualizar um recurso existente.

DELETE Apagar um recurso existente.

Esses quatro métodos, são as acções, apelidadas de verbos. Note-se que estes conceitos também estão presentes nas invocações remotas, sendo as classes, nomes e os métodos remotos, os verbos [Bor03].

3.2.4 Comparação

Antes de mais é necessário referir, que não se pode comparar o que não é comparável! Pode-se agrupar XML-RPC e SOAP, uma vez que, ambos são protocolos de transmissão de dados via *Internet* recorrendo ao XML e consequentemente estão sujeitos a comparações. REST é, quanto a si, uma filosofia que consiste nos princípios de utilização da Web, nomeadamente do protocolo HTTP e representa apenas uma alternativa às duas anteriores [Bor04]. A comparação que será feita nesta secção será entre uma arquitectura orientada a invocações remotas (XML-RPC e SOAP) e uma arquitectura orientada aos recursos (REST).

XML-RPC apresenta uma facilidade de utilização significativa, que é conseguida à custa das limitações da tecnologia, principalmente, no que diz respeito ao envio de informação complexa (objectos de negócio), questões de segurança, envio de documentos etc.. Os *web services* SOAP vieram precisamente remediar tais problemas. A tecnologia XML-RPC caiu em desuso, sendo apenas utilizada quando se trata da disponibilização de serviços elementares. A grande questão que se coloca actualmente, quando se pretende disponibilizar serviços para o exterior, é a da arquitectura a adoptar para implementação dos serviços. O debate incide geralmente sobre a utilização de SOAP ou REST.

Apesar destas duas arquitecturas apresentarem diferenças, vão de encontro ao mesmo objectivo — Usar os *standards* da *Internet* como HTTP, para veicular mensagens num formato específico, geralmente XML.

Os *web services* SOAP tem uma especificação densa e complexa. Além da especificação de base existem mais de 30 especificações complementares (WS-*), algumas das quais, ainda não estão terminadas.

3.3 Caso de Estudo : VIAPRE

Esta secção pretende evidenciar algumas das características dos *web services* referidas neste capítulo, através do relato da implementação de um *web service* desenvolvido ao longo deste projecto, com o intuito de solucionar um problema de integração entre as empresas VIAPRE e AXA. Será apresentado o problema, descritas as tecnologias e ferramentas usadas para a resolução do mesmo e serão analisados alguns detalhes de implementação julgados relevantes. Por fim, são apresentadas, algumas considerações acerca deste trabalho.

3.3.1 Problema

Aquando de um sinistro uma seguradora tem de averiguar e avaliar os prejuízos, sendo para tal feita uma peritagem. Seguradoras e empresas de peritagens são organizações distintas, isto é, não é da responsabilidade da seguradora efectuar peritagens, sendo esta, entregue a uma empresa especializada.

A VIAPRE é uma empresa de peritagens com 25 anos de experiência na área, possui uma aplicação Web para fazer toda a gestão de processo de peritagem, desde a inserção de uma nova requisição de peritagem até à conclusão da mesma. Actualmente, a inserção de uma requisição é feita manualmente na aplicação da VIAPRE de duas formas distintas:

- uma seguradora envia os dados de um sinistro via fax ou email, ficando, os funcionários da VIAPRE responsáveis pela inserção da requisição de peritagem
- certas companhias de seguros (como é o caso da AXA), possuem credenciais para aceder à aplicação da VIAPRE e introduzir directamente a requisição.

Seja qual for a estratégia usada para a inserção da requisição, requer intervenção humana. A seguradora AXA tem vindo a exercer pressões para que este processo seja automatizado. Sendo a AXA um cliente de elevada importância para a empresa de peritagens, a necessidade de integração é inevitável. Como se pode averiguar, as pressões externas referidas no Capítulo 2, justificam o esforço de integração.

A solução para este problema, passa pela implementação de um *web service* do lado da VIAPRE, que será invocável do lado da AXA e que permitirá a submissão automática de uma requisição de peritagem.

3.3.2 Ferramentas e Tecnologias

A escolha do tipo de *web services* orientou-se para o SOAP, por se tratar de uma tecnologia bem conhecida na empresa I2S e muito utilizada para solucionar vários tipos de problemas. A linguagem de programação escolhida foi o C#, sendo a plataforma de desenvolvimento o Visual Studio 2005 (VS2005).

Neste capítulo já se procedeu à apresentação da tecnologia e já foi comparada com outras alternativas para solucionar problemas da mesma natureza. Uma das desvantagens referidas na utilização deste tipo de *web services*, é a densidade da sua especificação e a sua consequente complexidade. Pretende-se demonstrar nesta secção que a implementação de um *web service* SOAP, não resulta numa tarefa de elevada complexidade, devido nomeadamente ao uso de *frameworks* e IDE's que facilitam a tarefa do programador.

O VS2005, propõe um *wizard* para criação de um projecto do tipo *ASP .NET Web Service Application*. O IDE cria desde logo, um *web service* muito básico, que não recebe argumentos e retorna a string “*Hello World*”. Pressionando F5 (comando que permite correr uma aplicação desenvolvida no IDE) tem-se o *web service* publicado e pronto a ser invocado.

A criação de um cliente para a invocação deste serviço, não apresenta maior complexidade do que a de criação do serviço. Muito rapidamente se cria um projecto Windows Form (poder-se-ia criar outro tipo qualquer de projecto³) através do *wizard* proposto pelo editor. O IDE cria desde logo uma interface muito básica. Do lado direito, no painel de propriedades, adiciona-se uma referência *web* para o *web service* previamente implementado. O VS2005, dispõe de um UDDI para encontrar *web services* e atribuir um nome para a referência. No cliente, pode-se agora desenhar um mecanismo para invocar o serviço. Sugere-se a criação de uma botão na interface para a invocação e criação de um área de texto para visualizar a resposta recebida. Muito rapidamente, se criam estes componentes recorrendo à *palette* que permite a criação de componentes de interfaces gráficas. Resta agora, instanciar um objecto que tem como nome o nome da referência ao *web service*, podendo-se, desde já, invocar um método com o mesmo nome que aquele foi criado do lado do *web service*.

Todo este processo não demora mais de dois minutos a concluir e testar. A ajuda fornecida pelo IDE ao programador, é considerável. O leitor ainda terá em mente as características que foram apresentadas acerca deste tipo de *web service*, nomeadamente a interface de invocação (WSDL) e o formato da mensagem SOAP, e que não foram evocados no processo de criação e invocação de um *web service* usando o VS2005. De facto, o editor trata de gerar a WSDL do serviço, assim como o código de invocação deste serviço (geração feita aquando da criação de uma *Web Reference* para o serviço do lado do cliente). Esta ferramenta permite que o programador apenas se preocupe com a lógica de negócio que o serviço precisa expor (quando se implementa o serviço) ou com o processamento da resposta de uma invocação remota (quando se implementa o cliente).

³Por exemplo, um projecto do tipo *Console Application*.

3.3.3 Considerações

O *web service* desenvolvido consiste apenas num método público *registraRequisicao*, que aceita como parâmetro de entrada uma string que representa o ficheiro XML que contém a informação necessária ao registo da requisição de peritagem e retorna também uma string com o seguinte formato:

- Requisição com ID xxxx criada com sucesso em dd-mm-aaaa:hh:mm:ss

O serviço é composto por outros métodos, mas que não estão visíveis para o exterior, são apenas métodos auxiliares para o registo de uma requisição de peritagem, e que de seguida se apresentam sumariamente.

- *validaXMLRecebido*: Permite validar de acordo com o seu *schema* o ficheiro XML recebido pelo *web service*.
- *gravaXMLRecebido*: Permite gravar fisicamente uma cópia de segurança do ficheiro recebido pelo *web service*.
- *criaRequisicao*: Permite criar um objecto do tipo *Requisicao* com os dados recebidos no ficheiro XML de entrada. Esse objecto que será mapeado para um *typed DataSet* e inserido na base de dados da VIAPRE.

A segurança é igualmente um aspecto significativo para este projecto. De facto, é essencial garantir que apenas a AXA poderá ter acesso ao serviço de requisição de peritagens. Para esse efeito, foi desenvolvida uma camada de segurança, mais precisamente de autenticação. A *framework* .NET na sua versão 2.0, tem em conta a questão da autenticação permitindo: *Basic Authentication*, *Windows Authentication* e autenticação personalizada. Neste caso é usada autenticação personalizada, que consiste na inserção de credenciais na mensagem SOAP, mais precisamente ao nível do *header* SOAP. A estratégia, passa por criar uma classe que deriva da classe *SoapHeader* (por exemplo, *AuthHeader*) e que deve conter dois campos públicos: *username* e *password*. A classe que implementa o serviço deve criar uma nova instância da classe *AuthHeader* e deve ser especificado, para cada método acessível do exterior, que este contém um *header* SOAP. Deste modo, cada pedido encapsula na mensagem SOAP as credenciais de quem requer o serviço [Nic04].

3.4 Conclusões

Este capítulo apresentou a tecnologia *web services* que não é mais do que a evolução das tecnologias de distribuição, tais como COM ou CORBA, colmatando as deficiências destas. Desde o seu nascimento, esta tecnologia teve detractores, que acusavam os *web*

services de não trazer nada de novo à Engenharia de *Software*, devido às fortes semelhanças com as tecnologias de distribuição [AG02], desde o paradigma de invocações remotas, passando pelo conceito de interface, até ao serviço de directório. No entanto há uma novidade, que se prende com a introdução de *standards*, sendo esta, a única forma de se garantir interoperabilidade. Por outro lado, as suas extensas especificações (WS-*), suscitam críticas à tecnologia, acusando-a a de obter fracos desempenhos quando comparada com as tecnologias de distribuição [WRC06]. A sua extensa especificação não implica difícil implementação, já que existem *frameworks* e IDE's que facilitam o desenvolvimento de *web services*, sendo que, o programador apenas tem de se preocupar com a lógica de negócio a expor (se implementar o serviço) ou o tratamento da resposta (se implementar um consumidor do serviço), ficando a definição da interface (WSDL), a construção da mensagem SOAP e o código de invocação a cargo da ferramenta utilizada.

Após estas observações sobre os *web services* SOAP (tema central deste capítulo), é relevante realçar a popularidade que os *web services* REST têm vindo a gozar. Com uma especificação muito menor do a dos *Big Services*, esta tecnologia apresenta-se, como uma alternativa altamente viável, o que tem originado vários debates que opõem os defensores de cada abordagem.

Se esta tecnologia obtém um óptimo grau de adequação à resolução de ligações ponto-a-ponto, o mesmo não se aplica a cenários em que a arquitectura de integração é composta por várias aplicações em que, rapidamente, a gestão da mesma se torna insustentável. Este último facto, justificou esforços e investigações para solucionar o problema, até se chegar a uma nova tecnologia⁴: *Enterprise Service Bus* que será analisada em detalhe no próximo capítulo.

⁴O aparecimento dos *Enterprise Service Bus* não significa o desaparecimento dos *web services*.

Capítulo 4

Enterprise Service Bus

O *Enterprise Service Bus* (ESB) é uma tecnologia informática, que tem vindo a gozar de uma popularidade crescente. O seu principal objectivo é a integração de aplicações empresariais que inicialmente não foram pensadas para interagir. Segundo Roy Shuttle, do grupo Gartner inc, um ESB é uma nova arquitectura que explora *web services*, os *message oriented middleware* (MOM) e o *routing* inteligente, assim como a transformação de mensagens [Wik09c]. Este capítulo debruça-se sobre esta recente tecnologia como solução para problemas de integração. Antes de mais, é feita uma apresentação global da tecnologia e é de seguida, dada maior ênfase ao paradigma de troca de mensagens, usado para responder a problemas de integração de software. Finalmente, é relatada a implementação de várias provas de conceito, desenvolvidas com intuito de averiguar a adequação desta tecnologia ao problema de integração entre mediadores e companhias de seguros.

4.1 Introdução ao *Enterprise Service Bus*

Esta secção pretende incutir ao leitor uma visão global da tecnologia *Enterprise Service Bus*, enquadrando-a primeiramente, no contexto das arquitecturas orientadas aos serviços e com as *Enterprise Application Integration*, que foram os predecessores dos *Enterprise Service Bus*.

4.1.1 Enquadramento

É recorrente, na literatura, associar *web services* e arquitecturas SOA. De facto, os *web services* fazem parte das tecnologias de referência para se implementar SOA. A problemática das ligações ponto-a-ponto, relacionada com a difícil gestão da arquitectura,

contribuiu para o aparecimento de outras tecnologias, como os *Enterprise Application Integration*.

Note-se, que o aparecimento de novas tecnologias não significa o desaparecimento dos *web services* que continuam a ter um papel importante na integração de aplicações empresariais. Antes de mais, define-se sumariamente em que consiste SOA e de seguida, apresenta-se a tecnologia *Enterprise Integration Application*.

4.1.1.1 *Service-Oriented Architecture*

Em termos gerais, uma arquitectura orientada aos serviços (que daqui em diante será apenas referenciada por SOA) consiste numa arquitectura de integração de software que assenta no conceito de serviço [Bal05]. Esta arquitectura pode ser igualmente vista como uma colecção de serviços que comunicam entre si. A comunicação envolve frequentemente passagem de dados. A comunicação pode ocorrer entre dois ou mais serviços que coordenam a execução de determinada actividade. A palavra-chave destas arquitecturas prende-se com o conceito de serviço, que é uma função bem definida, auto-contida (*self-contained*) e que não depende do contexto ou do estado de outros serviço para executar [Bar].

As primeiras arquitecturas a corresponder à definição acima apresentada, foram as arquitecturas baseadas nas tecnologias DCOM e CORBA, já referidas nesta dissertação. Com a evolução da tecnologia, os *web services* tornaram-se os actores principais das arquitecturas orientadas aos serviços.

Durante algum tempo, a SOA esteve fortemente ligada aos *web services*, cometendo-se por vezes o erro de dizer “os *web services* são uma arquitectura orientada aos serviços”, o que é obviamente falso. Os *web services* são, apenas, uma tecnologia que favorece a implementação de arquitecturas do tipo SOA. Sendo estes, uma evolução das invocações remotas (RPC) e consequentemente caracterizados por ligações ponto-a-ponto, têm o inconveniente de se tornar rapidamente difíceis de manter quando, o número de intervenientes na arquitectura de integração tende a aumentar. Para encarar essa problemática de gestão, investiram-se esforços na elaboração de outras topologias, como sucessoras das tradicionais ligações ponto-a-ponto, daí o nascimento da topologia *Hub-and-Spoke* e o aparecimento dos primeiros produtos de Software ditos, *Hub-and-Spoke*.

4.1.1.2 *Enterprise Application Integration*

Em primeira instância, uma *Enterprise Application Integration* é uma arquitectura de *software* que permite a troca de informação entre aplicações heterogéneas e por extensão, dá-se o mesmo nome a um sistema de *software* que permite a realização desta arquitectura, implementando os fluxos de informação entre as diversas aplicações envolvidas

Enterprise Service Bus

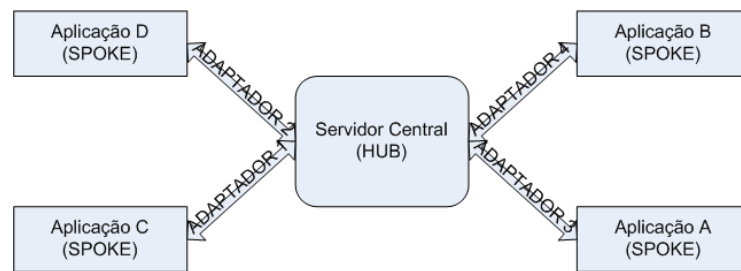


Figura 4.1: Arquitetura *Hub-and-Spoke*

na arquitetura [Bes]. Estes produtos de software seguem uma arquitetura *Hub-and-Spoke* como ilustrado na Figura 4.1, isto é, consistem num servidor centralizado (*hub*), que recebe informação de todas as aplicações envolvidas na integração, transforma-a e re-encaminha essa informação para outras aplicações (*spokes*). De seguida apresentam-se sumariamente, as características gerais de tal software [Far07].

adaptadores que fazem a introspecção do modelo de dados e dos serviços de uma aplicação, de modo a verificar se há alterações que devam ser expostas às outras aplicações do EAI.

middleware assíncrono responsável pela garantia de entrega assíncrona das mensagens trocadas no seio da EAI;

gestor de serviço capaz de aplicar serviços tais como, transformação, *routing*, filtragem de mensagens

consola de administração capaz de supervisionar os componentes técnicos do EAI assim como os serviço de negócio, incluídos na solução de EAI.

A maior parte dos fornecedores propõe a pilha de *middleware* acima referida e a riqueza funcional não é problema para as empresas que adoptam esta solução. Note-se que, de tanta funcionalidade, advém uma complexidade significativa, da qual advêm as seguintes desvantagens.

preço excessivo de licenças devido à riqueza funcional destas soluções, o seu preço é relativamente elevado.

custo inicial da integração de todas as aplicações é elevado.

escalabilidade esta solução é pouco escalável, uma vez que o *hub* agrega o tráfego de mensagens, de todas as aplicações.

tolerância a falhas dado que todas as mensagens passam pelo *hub*, este constitui um ponto único de falha (*single point of failure*).

<i>Standard</i>	Descrição
JMS	<i>routing</i> de mensagens dentro do bus
SOAP	Formato da mensagem enviada a um <i>web service</i>
WSDL	Interface de acesso a um <i>web service</i>
XML	Formato universal de representação de informação estruturada
XSL	Linguagem para transformação de mensagens
XPath	Linguagem de interrogação a mensagens XML (usado para <i>routing</i>)
BPM	Gestão de processos de negócio

Tabela 4.1: *Standards* Usados nos ESB's

Face ao carácter proprietário destas soluções adivinham-se problemas de interoperabilidade entre duas soluções de EAI. O seu custo elevado e os problemas mencionados, inerentes à própria arquitectura, são um entrave para muitas empresas em adoptar este tipo de solução.

4.1.2 *Enterprise Service Bus* — Definição

Os *Enterprise Service Bus* (ESB) podem ser vistos como a 2ª geração de *Enterprise Integration Application*. A principal diferença prende-se com a topologia em *bus*, distribuindo a lógica de integração ao longo deste, ao contrário do que acontecia com os EIA, que tinha a lógica de integração centralizada num *hub*.

Um ESB pode ser encarado de três formas distintas [Wik09c]:

produto de software dum ponto de vista técnico, um ESB é uma “peça” de software de *middleware*, uma vez que reside entre as várias aplicações a integrar.

conjunto de produtos de software por extensão ao ponto de vista anterior. É legítimo afirmar que essa “peça” de software é constituída por “peças” menores, uma vez que os ESB's fazem uso de *standard* de modo a garantir máxima interoperabilidade. Na tabela 4.1 apresentam-se alguns dos *standards* encontrados na maioria dos produtos ESB's.

construtor de arquitectura dum ponto de vista conceptual. O *Enterprise Service Bus* pode ser visto como uma infraestrutura, que permite a implementação de uma arquitectura (muitas das vezes SOA) baseada no conceito de *bus*, o que permite reduzir o número de ligações ponto-a-ponto, aumentando o grau de flexibilidade.

Um ESB pretende basear-se em *standards*, ser flexível e suportar vários transportes. Note-se que não são novos produtos, apenas uma nova forma de integrar aplicações, coordenar recursos e manipular informação. Em si, os ESB's não são uma arquitectura SOA, mas permitem, implementar uma arquitectura orientada aos serviços [Bal05] [PT04]. Um

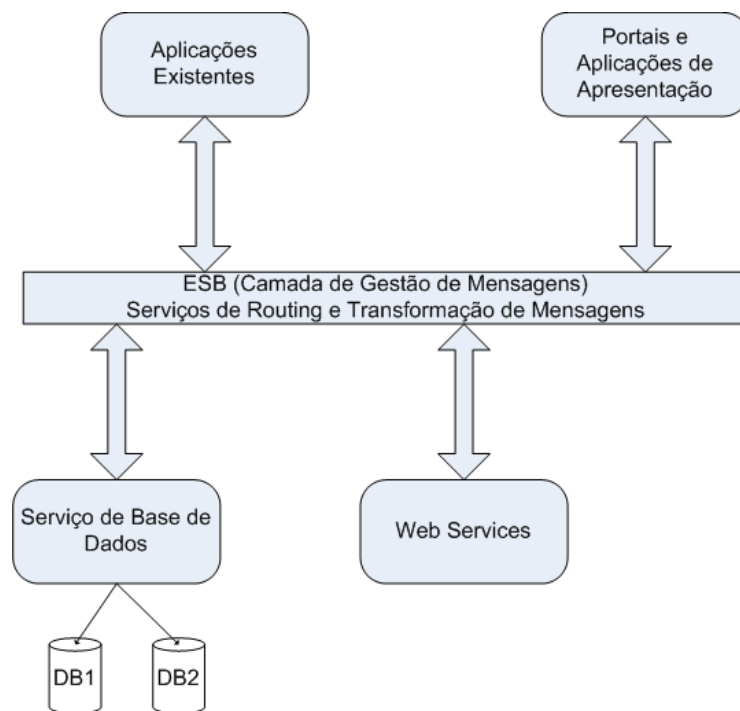


Figura 4.2: Arquitectura em *Bus*

bus não tem necessariamente que possuir o conceito de serviço, apresentado nesta secção. A Figura 4.2, ilustra uma possível arquitectura de integração usando um ESB.

As principais vantagens inerentes a esta tecnologia prendem-se com a gestão das mensagens que circulam no *bus*. De facto, as grandes funcionalidades dos ESB's prende-se com o *routing* inteligente, a transformação de mensagens, ou a filtragem de mensagens.

Na secção seguinte apresentam-se, com maior detalhe, esses conceitos chave, desta promissora tecnologia.

4.2 Paradigma: Troca de Mensagens

Este paradigma já foi evocado no Capítulo 2 e apresenta-se como um conceito incontornável quando estão em causa problemas de integração. Os *Enterprise Service Bus*, assentam precisamente nesse conceito, como já foi referido na Secção 4.1. Cada aplicação que deseja comunicar com outra aplicação ligada ao *bus*, envia-lhe uma mensagem e este encarrega-se de a entregar.

Este paradigma abarca um conjunto de conceitos relacionados, também designados por padrões de integração empresarial (*Enterprise Integration Patterns* — EIP) e cuja apresentação é feita de seguida.

4.2.1 Processamento de Mensagens

Os ESB's veiculam a informação ao longo do *bus*, podendo esta ser: transformada, filtrada, reencaminhada e processada. As filtrações, transformações acontecem nos *end-points*, quer de entrada quer de saída, ao contrário do processamento que é efectuado nos componentes.

Os componentes são um conceito chave na *framework* Mule ESB¹, que permitem o desenvolvimento de aplicações empresariais segundo uma arquitectura em *bus*. É neles, que o programador coloca a lógica de negócio. Uma das características deste componente é que não precisa de ter nenhum código específico à *framework*, podendo ser:

- Java POJO's
- Spring Bean
- Web service

Geralmente, os componentes são específicos a cada aplicação, pelo que são desenvolvidos pelo programador e dependem da lógica de negócio que se pretende expor. No entanto, existem componentes genéricos que podem ser usados:

- Log Component: Componente que regista no sistema de *log*, a requisição do serviço;
- Echo Component: Componente que retorna exactamente o que recebe;
- Bridge Component: Componente que não efectua processamento à mensagem. Despacha directamente para o *endpoint* de saída;
- Null Component: Componente que coloca o *payload* a *null*.

4.2.2 Filtragem de Mensagens

Por vezes é necessário filtrar a informação. Os ESB's são muitas vezes caracterizados pela sua capacidade fazer *routing* inteligente de mensagens. Essa capacidade advém da possibilidade de se poder filtrar a informação nos *endpoints* e consoante essa informação reencaminhar a mensagem para um determinado componente ou *router* de saída. Existem vários tipos de filtros aplicáveis à mensagem que é recebida num *endpoint*. Quando aplicados à entrada esses filtros permitem determinar se interessa que a mensagem seja processada ou descartada. Quando aplicados à saída, os filtros permitem determinar para onde será despachada a mensagem. Existem vários tipos de filtros que se enumeram de seguida:

¹Página Oficial disponível em : <http://www.mulesource.org>

- Filtro baseado no tipo do *payload*
- Filtros baseados em expressões
 - Expressões XPath
 - Expressões OGNL
- Filtros baseados em expressões regulares
- Filtros baseados em *WildCards*
- Filtros baseados no tipo de exceção
- Filtros baseados em mensagens de propriedades
- Filtros lógicos
 - Filtro And
 - Filtro Or
 - Filtro Not

4.2.3 Transformação de Mensagens

A transformação de mensagens é um dos conceitos centrais dos ESB's. De facto, nem todas as aplicações ligados ao *bus* entendem o mesmo formato de mensagens. Assim sendo, pode ser necessário transformar a mensagem consoante a aplicação que a tratará. Essas transformações ocorrem nos *endpoints*. No caso de um *endpoint* de entrada, a mensagem é transformada antes de ser enviada ao componente. No caso de se tratar de um *endpoint* de saída, a transformação ocorre mesmo antes da mensagem ser despachada para outro ponto terminal.

Refira-se no entanto, que os transformadores são da responsabilidade do programador, como foi mencionado no parágrafo acima, mas são também da própria *framework*. Imagine-se o seguinte cenário: uma encomenda é sempre representada no formato XML, no entanto, pode chegar ao *bus* por intermédio de vários transportes, dependendo da aplicação que criou a encomenda (por exemplo, HTTP ou JMS). Desta forma, como saberá o componente ler a mensagem, uma vez que são independentes do formato da mensagem como foi referido na subsecção 4.2.1. A resposta é que são aplicadas transformações no *router* de entrada, mesmo antes que este entregue a mensagem ao componente para processamento. Todas essas transformações são transparentes para o programador.

4.2.4 Routing Inteligente de Mensagens

Um dos conceitos principais da tecnologia referida neste capítulo é a capacidade de encaminhar mensagens ao longo do bus. Existem dois aspectos do *routing* nos ESB's que convém mencionar: os pontos terminais (*endpoints*) e os *routers*.

Os *endpoints* têm por missão interligar serviços. Fundamentalmente, um *endpoint* é um canal específico através do qual um determinado componente pode enviar mensagens pelo seu *endpoint* de saída (*outbound endpoint*) a outro componente que receberá essas mensagens num dos seus *endpoints* de entrada, (*inbound endpoint*).

Os *routers*, são usados para controlar a forma como as mensagens são recebidas e despachadas. Um *router* (de entrada) controla e manipula a mensagem antes desta ser fornecida ao componente para processamento, podem filtrar, agregar ou re-sequenciar mensagens de chegada. O *router* de saída consegue fazer o mesmo, mas desta vez após o processamento da mensagem, antes de esta ser despachada para outro componente.

4.3 Caso de Estudo: Provas de Conceito

Após o estudo da tecnologia *Enterprise Service Bus*, julgou-se necessário pôr em prática os conceitos adquiridos, de modo a ficar-se com maior percepção do impacto desta tecnologia na solução do problema de integração entre mediadores e companhias de seguros.

4.3.1 Mule ESB

A escolha tecnológica do ESB, recaiu no Mule ESB uma vez que é uma tecnologia livre, que está actualmente a ter um grande sucesso (existem na *Internet* casos de sucesso, de resolução de problemas de integração, recorrendo a esta *framework*) e que está a ser usada dentro da empresa I2S, para outros projectos.

4.3.1.1 Framework

Mule ESB é uma *framework* leve, muito recente, desenvolvida em Java, que implementa um *Enterprise Service Bus* e consequentemente permite a interligação de várias aplicações. O desenvolvimento da *framework* foi baseado nos *Enterprise Integration Patterns*, alguns dos quais, já foram discutidos neste documento. Existem dois aspectos fundamentais inerentes a esta *framework*, sendo eles, a implementação em Java dos componentes, ou transformadores e filtros customizados e a configuração, que é feita através de um ficheiro XML. É neste ficheiro que são declarados os serviços (componentes, *endpoints*), os transformadores e os filtros a aplicar. A *framework* faz uso de software de

terceiras partes apresentadas com mais detalhe na subsecção 5.2, como por exemplo (a lista apresentada, não é exaustiva):

- Spring Framework
- Apache Axis
- Apache CXF
- Apache WSS4J
- ACEGI Security
- JAAS

A versão da distribuição usada é a v2.0.2, sendo usado para auxiliar o desenvolvimento, o IDE eclipse 3.2 (Europa).

4.3.1.2 Mule IDE

Mule IDE é um *plugin* do eclipse cujo objectivo é facilitar a configuração de aplicações desenvolvidas com base no Mule. Permite uma visualização e configuração gráfica da arquitectura de integração, através da tecnica *Drag-and-Drop*.

4.3.2 Workflow Básico no Bus

A aplicação Gis Mediadores é uma aplicação do tipo *Windows Form*, desenvolvida em C++, alguns dos módulos associados foram desenvolvidos usando C#. As seguradoras, muitas das quais são clientes da I2S, têm as suas aplicações alojadas no WebSphere Application Server — WAS.

Após a leitura e estudo dos conceitos básicos desta *framework* através do *User Guide* e dos exemplos fornecidos junto com a distribuição, julgou-se necessário implementar o cenário que consiste no desenvolvimento do circuito básico de integração: isto, é, a partir de um cliente desenvolvido, com recurso à *framework* .NET, invocar um serviço no Mule que por sua vez invoca um *web service* no WAS; a resposta tem obviamente de ser devolvida à aplicação .NET, responsável pela primeira invocação, como ilustrado pela Figura 4.3.

Julga-se digno de relevo salientar com maior detalhe as características do modelo desenvolvido. O componente referenciado pela figura 4.3, é um *web service* exposto no Mule e desenvolvido com a tecnologia Axis. A funcionalidade deste componente é retornar o que entra em parâmetro, após ter concatenado outra *string*, isto é, o serviço contém um método com a seguinte assinatura:

```
String echo(String s)
```

Enterprise Service Bus

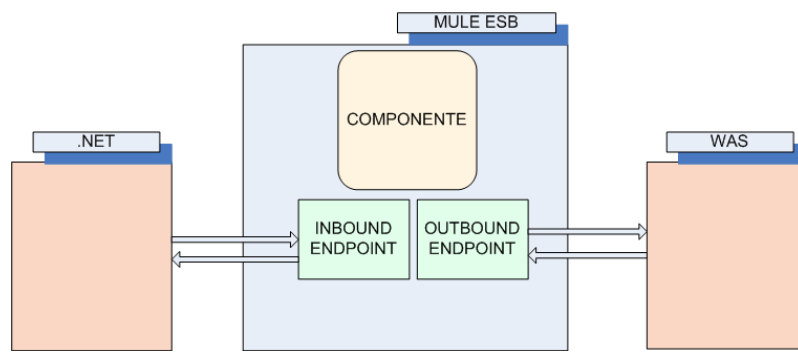


Figura 4.3: *Workflow* Básico dentro do *Bus*

que apenas retorna a String que recebeu e adiciona-lhe Mule -> de modo a, no final poder verificar-se por onde passou a string. A sequência deste cenário é a seguinte:

1. Cliente .Net invoca o *web service* com o seguinte parâmetro .NET ->
2. Componente no Mule acrescenta Mule -> à String que recebeu e invoca o *web service* no WAS (com a mesma funcionalidade)
3. WAS acrescenta WAS -> à string recebida e devolve a resposta (.NET -> Mule -> WAS) ao Mule, que por sua vez devolve a resposta ao cliente .NET, que finalmente acrescenta .NET à string. Note-se que a string que circula do .NET até ao WAS adopta o nome de *payload*.

O cenário foi rapidamente implementado com sucesso. Foi feita uma ligeira alteração, na medida em que se aplicou uma transformação à mensagem recebida no Mule em proveniência do WAS. Uma invocação na aplicação .NET com o seguinte parâmetro; .NET -> produz a seguinte resposta : .NET -> Mule -> WAS -> Mule (again) -> .NET

4.3.3 Circuito Básico com Segurança Activa

Em termos genéricos, pode afirmar-se que já se conseguiu integração, na medida em que, a partir da aplicação dos mediadores (simulação) se consegue invocar um serviço remoto, recorrendo para tal a um *Enterprise Service Bus*, que age como intermediário. É agora tempo, de atender às questões de segurança, mais precisamente, autenticação.

Uma das especificações WS-* dos *web services* SOAP (WS-Security) consiste num protocolo de comunicação que providencia um meio de se aplicar segurança a *web services*[Nad06]. Mais uma vez, nesta dissertação, se reforça a ideia que o uso de *standards* é essencial para solucionar problemas de integração, e particularmente quando as aplicações em questão (como é o caso nestas provas de conceito), são desenvolvidas com recurso a linguagens de programação distintas. O Wse 3.0 é a implementação da especificação WS-Security para aplicações desenvolvidas com base na *framework* .NET. Do

lado da linguagem Java (usada dentro do *bus* e nas companhias), a *framework* WSS4J, encarrega-se da segurança segundo a especificação do OASIS antes referida.

Neste cenário, a principal preocupação prende-se com a autenticação. A estratégia seguida faz uso de *security tokens* e consiste em enviar o nome de utilizador e password na própria mensagem SOAP. As credenciais serão validadas junto de uma fonte de autenticação, como por exemplo *Active Directory* ou uma base de dados relacional. A *framework* Mule, usa a *framework* Apache CXF como motor de *web services*, que permite a configuração de segurança. No WebSphere, apenas existe suporte para Axis. Ambos os motores, usam a *framework* WSS4J para a implementação da segurança segundo a especificação WS-Security. De seguida descreve-se como é conseguida a implementação de autenticação, do lado da aplicação cliente.

1. Criação de um objecto *UsernameToken* que tem como atributos o *username* e *password* e a forma como devem ser enviadas as credenciais (*plain text* ou *hashed*).
2. Criação de uma política de segurança, através da instanciação de um objecto *Policy*. Adiciona-se, posteriormente, a essa política a seguinte asserção *UsernameOverTransportAssertion*. Finalmente atribui-se a política criada à instância do *web service*.
3. O *web service* de acesso ao *bus*, está pronto a ser invocado. A listagem 4.1, retrata o que foi acima referido.

```

1 muleProxyWS.MediadoresServiceWse cxfWithSecurity =
2     new muleProxyWS.MediadoresServiceWse();
3
4
5 WindowsIdentity id =
6     WindowsIdentity.GetCurrent();
7
8
9 UsernameToken token =
10    new UsernameToken(username, id.User.Value, PasswordOption.SendPlainText);
11 cxfWithSecurity.SetClientCredential(token);
12
13 Policy clientPolicy = new Policy();
14 clientPolicy.Assertions.Add(new UsernameOverTransportAssertion());
15
16 cxfWithSecurity.SetPolicy(clientPolicy);

```

Listing 4.1: Excerto de código de invocação com WS-Security

Resta neste momento, configurar a segurança no Mule e no WebSphere. Note-se, que não é descrita a configuração da segurança no WebSphere, para não sobre carregar esta secção, e por não se julgar digno de relevo, para a compreensão dos ESB's. Não existem

grandes entraves à configuração da segurança usando Axis e WSS4J no servidor de aplicação WAS. Os manuais disponibilizados pela Apache, exemplificam bem como se pode implementar segurança. Segue, na listagem 4.2, um excerto do ficheiro de configuração do *bus*, relativo à configuração do *endpoint* de entrada e aos *Spring Beans* necessários para a configuração da autenticação, baseada em ACEGI e JAAS.

```

1  <spring:bean id="ACEGIAuthCallback"
2      .
3      .
4  </spring:bean>
5      .
6      .
7  <spring:bean id="authenticationManager"
8      .
9      .
10 </spring:bean>
11      .
12      .
13 <spring:bean id="JAASAuthenticationProvider"
14      .
15      .
16 </spring:bean>
17      .
18      .
19 <inbound>
20     <cxfr:inbound-endpoint address="http://localhost:88/mediadores"
21         remoteSync="true" remoteSyncTimeout="5000"
22         responseTransformer-refs="muleappender">
23
24         <cxfr:inInterceptors>
25             <spring:bean
26                 class="org.apache.cxf.binding.soap.saaj.SAAJInInterceptor" />
27             <spring:bean
28                 class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
29                 <spring:constructor-arg>
30                     <spring:map>
31                         <spring:entry key="action" value="UsernameToken_Timestamp" />
32                         <spring:entry key="passwordType" value="PasswordText" />
33                         <spring:entry key="passwordCallbackRef">
34                             <spring:ref bean="ACEGIAuthCallback" />
35                         </spring:entry>
36                     </spring:map>
37                 </spring:constructor-arg>
38             </spring:bean>
39         </cxfr:inInterceptors>
40     </cxfr:inbound-endpoint>
41 </inbound>

```

Listing 4.2: Configuração de segurança no *endpoint* de entrada no Mule

Note-se a declaração do atributo `<cxfr:inInterceptors>` na linha 24. Tal como o seu nome sugere, este componente intercepta a mensagem recebida, trata de extrair as credenciais e de as validar. Em caso de sucesso, a mensagem é despachada para o serviço e posteriormente processada. Em caso de falha, é devolvida uma excepção informando sobre o erro ocorrido.

Uma vez a mensagem SOAP interceptada, é necessário definir a estratégia de autenticação. Veja-se na linha 34 a declaração de uma rotina de *callback*. É comum, a implementação dessa rotina ficar a cargo do programador, sendo nela especificada a fonte de autenticação (base de dados ou LDAP por exemplo) e as configurações necessárias para se aceder a essas fontes. Inicialmente seguiu-se essa abordagem, mas rapidamente se concluiu que não era satisfatório desenvolver uma rotina de autenticação caseira, quando já existem *frameworks* que tratam eficazmente desse assunto.

A *framework* ACEGI Security, desenvolvida em 2004, fornece mecanismos avançados de autenticação, autorização e outros aspectos relacionados com segurança para aplicações J2EE baseadas na *framework* Spring, fornecendo na sua distribuição uma série de rotinas de *callback* para propósitos de autenticação [Ale04]. No âmbito desta prova de conceito, foram usadas duas fontes de autenticação: LDAP e autenticação junto do sistema operativo. A segunda apresenta-se apenas como uma alternativa à primeira, quando esta não puder ser aplicada. De facto, nem todos os mediadores terão um directório LDAP para questões de autenticação (nesses casos, será usada a autenticação do sistema operativo). Admite-se que o *bus* reside numa máquina (do lado dos mediadores) sendo necessário ter uma conta no sistema operativo dessa máquina, para invocar os serviços do *bus*.

Em suma, a autenticação do utilizador fica a cargo da *framework* ACEGI, que autentica o utilizador junto de um directório LDAP, se este existir no mediador, no caso contrário, adoptar-se-á outra abordagem: autenticação junto do sistema operativo. Essa abordagem também faz uso da *framework* ACEGI, que delegará à *framework* JAAS, outra *framework* que atende questões de autenticação e autorização e que dispõe de um módulo *NTLoginModule* que permite a autenticação junto do sistema operativo (entenda-se Windows, existem também módulos para sistemas UNIX), como referido anteriormente.

O *bus* que é servidor para a aplicação dos mediadores, é o cliente do servidor de aplicação (isto, é, dos *web services* expostos pela companhia). A configuração do *outbound endpoint* é semelhante à do *inbound endpoint*, que se apresentou anteriormente, pelo que não será apresentada detalhadamente. Salienta-se apenas a necessidade de declaração de um interceptor, ao qual são passadas as credenciais, como se pode ver na listagem 4.4. Note-se que nesta listagem, a password é definida através de uma rotina de *callback*, mas poderia também ser declarada directamente neste ficheiro de configuração.

```
1 <cxfr:outbound-endpoint
2     address="http://localhost:9080/WebServiceProject/services/WebSphereWS"
```

```

3      name="WebSphereWSService"
4      clientClass="com.i2s.websphere.ws.WebSphereWSService"
5      wsdlLocation="http://localhost:9080/services/WebSphereWS?wsdl"
6      wsdlPort="WebSphereWS"
7      operation="echo"
8      synchronous="true">
9
10     <cxfr:outInterceptors>
11         <spring:bean
12             class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor" />
13         <spring:bean
14             class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
15             <spring:constructor-arg>
16                 <spring:map>
17                     <spring:entry key="action" value="UsernameToken" />
18                     <spring:entry key="user" value="i2suser" />
19                     <spring:entry key="passwordType" value="PasswordText" />
20                     <spring:entry key="passwordCallbackRef"
21                         value-ref="clientCallback" />
22                 </spring:map>
23             </spring:constructor-arg> </spring:bean>
24     </cxfr:outInterceptors>
25 </cxfr:outbound-endpoint>

```

Listing 4.3: Configuração de segurança no *endpoint* de saída no Mule

4.3.4 *web services*Proxy

Conseguiu-se, a partir de uma aplicação .NET invocar um *web service* no Mule, que por sua vez o encaminha para um *web service* alojado no WebSphere de forma segura, autenticando a entidade que requisita o serviço. Obviamente, a questão da segurança foi apenas parcialmente resolvida, uma vez que as credenciais circulam em claro pela rede. Sabe-se que a especificação WS-Security oferece mecanismos para encriptar e/ou assinar a informação circulante. Todavia, existe também a possibilidade de usar um canal de comunicação seguro, como é o HTTPS. É de salientar o facto de que se consegue construir o circuito básico de integração, implementando apenas uma única classe Java no *bus*: a classe que contém a lógica de negócio do serviço, que para esta prova de conceito consiste apenas em retornar o que é recebido em parâmetro.

É importante realçar a função do *web service* do Mule, que serve de ponto de entrada para o *bus*. Este serviço não pretende expor nenhuma lógica de negócio, apenas serve para receber o pedido dos mediadores e consoante esse pedido encaminhar para um determinado serviço na companhia. Num cenário básico, a aplicação dos mediadores requisita

um serviço ao Mule, invocando o seu *web service* de entrada, com um determinado conjunto de parâmetros e este, por sua vez, encaminha o pedido tal como o recebeu². Num cenário mais complexo, o Mule poderá, por exemplo, ter que extrair os parâmetros que recebeu dos mediadores, transformá-los, invocar um *web service* com alguns desses parâmetros e de seguida invocar outro com o resultado do primeiro e mais alguns parâmetros que recebeu dos mediadores.

Já foi demonstrado que a tecnologia ESB permite a transformação e *routing* inteligente de mensagens. Uma abordagem rigorosa, revela que a implementação da classe que suporta o *web service* de entrada do Mule é desnecessária. Pretende-se apenas que o ponto de entrada no *bus* tenha a capacidade de extrair e transformar os dados recebidos da aplicação dos mediadores, de forma a poder encaminhar devidamente os pedidos. Em suma, o objectivo deste serviço é fazer *proxy* de *web services*.

O Mule tem essa funcionalidade, que pode ser usada em conjunto com o motor de *web services* CXF [Mul08]. A listagem 4.4, ilustra a configuração de *proxy* de *web services*.

```

1 <inbound>
2   <cxf:inbound-endpoint address="http://localhost:8081/my/service"
3     wsdlLocation="wsdl/entry.wsdl"
4     serviceName="EntryService"
5     proxy="true">
6 </cxf:inbound-endpoint>
7 </inbound>

```

Listing 4.4: Configuração de segurança de um *proxy* de *web services*

Normalmente, quando se implementa *web services*, é feito o *bind* da mensagem SOAP para POJO's. Por exemplo, um *web service* pode receber um objecto *Encomenda* e processá-lo directamente ao nível da linguagem de programação e de seguida despachá-lo para outro serviço. Neste caso, ao especificar o atributo *proxy* na linha 5, o *payload* passa a ser o próprio *body* da mensagem SOAP pelo que, para extrair dados ou invocar serviço do WebSphere, é necessário aplicar transformações XSLT ao *payload*.

Surgiram dificuldades na implementação desta prova de conceito uma vez que, quando se especifica que se pretende fazer *proxy* de *web services*, o Mule publica uma WSDL demasiada genérica que descreve um método *invoke* que apenas recebe como parâmetro um objecto do tipo *Object*. Gastou-se muito tempo na publicação de uma WSDL “caseira”, mas sem sucesso.

Decidiu-se então submeter este problema à comunidade Mule através do seu *forum*. A resposta que foi dada, é que a distribuição que se estava a usar continha alguns *bugs* que seriam corrigidos na próxima distribuição. Nessa altura, suspenderam-se os desenvolvimentos no Mule para se poder dar início à escrita desta dissertação.

²Entenda-se com os mesmos parâmetros

4.4 Conclusões

Os *web services* são uma tecnologia adequada à resolução de problemas de integração incluindo ligações ponto-a-ponto. Quando a arquitectura de integração envolve um número considerável de aplicações, a gestão das ligações torna-se complexa. Os *Enterprise Integration Application* sugeriram precisamente para solucionar este problema, na medida em que cada aplicação da arquitectura de integração comunica com um servidor central que encaminha informação para as aplicações de destino. Por sua vez esta tecnologia, também apresenta limitações, o que coloca entraves à sua utilização. Os *Enterprise Service Bus* podem ser vistos como a 2ª geração de EAI e solucionam os problemas relativos aos EAI.

Foi dada neste capítulo, uma visão global desta tecnologia, evidenciando a sua filosofia que assenta no paradigma de troca de mensagens (discutido no Capítulo 2), assim como as suas principais características, tais como *routing* inteligente de mensagens, a filtragem de mensagens e a transformação de mensagens.

Através da implementação de algumas provas de conceito, afirma-se que o aparecimento dos ESB's não implica o desaparecimento dos *web services*. De facto, estas duas tecnologias podem ser conjugadas para solucionar problemas de integração. Os *web services* apresentam-se como uma interface de acesso universal ao *bus*, como ilustrado nas provas de conceito referidas neste documento. Note-se, no entanto, que podem ser usadas outras tecnologias como interface de acesso ao *bus*, como é o caso da tecnologia JMS mas que, ao contrário dos *web services*, não garante interoperabilidade, uma vez que só apresenta compatibilidade com aplicações escritas em Java.

Capítulo 5

Arquitectura de Integração

Após o estudo das duas tecnologia possíveis para a resolução do problema de integração entre mediadores e companhias, é tempo de se proceder a uma comparação das arquitecturas providenciadas pelas duas. Primeiramente, é justificada a utilização de um ESB como possível solução. De seguida é feita uma revisão tecnológica relativa a *web services* e ESB's. Finalmente, analisa-se como cada tecnologia satisfaz os requisitos apresentados no Capítulo 2.

5.1 Justificação do estudo da tecnologia *Enterprise Service Bus*

O problema de integração entre mediadores e companhias de seguros, enunciado no Capítulo 1, refere que os mediadores pretendem obter determinada informação em tempo real. A abordagem mais óbvia, consiste em disponibilizar via *web services* a informação a que os mediadores pretendem ter acesso, sendo que estes apenas têm de invocar essas funcionalidades directamente a partir da aplicação Gis Mediadores. A arquitectura de integração resume-se, neste caso, a um conjunto de ligações ponto-a-ponto entre mediadores e companhias de seguros, como sugere a Figura 5.1.

Em primeira análise, a utilização de um *Enterprise Service Bus* como arquitectura de integração pode resultar despropositada, quando esta arquitectura envolve apenas duas aplicações e pode consequentemente ser implementada através de ligações ponto-a-ponto. De facto, o intuito desta tecnologia é solucionar o problema de gestão das ligações ponto-a-ponto quando o número de aplicações envolvidas no processo de integração tende a ser significativo. No entanto, a utilização de um *bus* como solução ao problema de integração entre mediadores e companhias de seguros é uma clara alternativa às ligações ponto-a-ponto pelos motivos que se descrevem de seguida:

Arquitectura de Integração

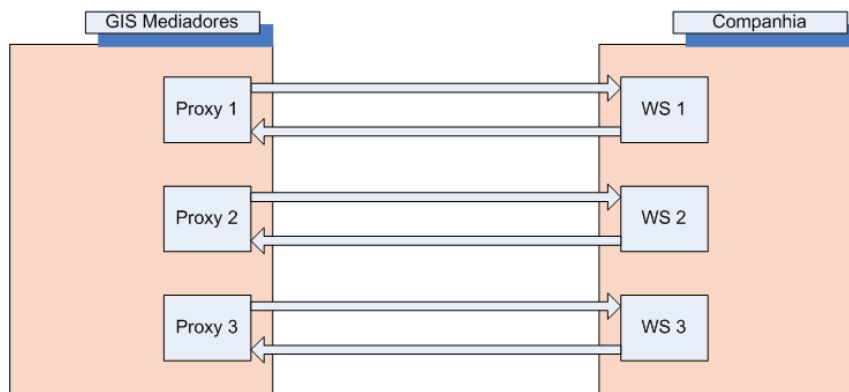


Figura 5.1: Arquitectura de Integração — Ligações Ponto-a-Ponto

Ausência de código uma vez que, os ESB's já foram apresentados no Capítulo 4, como sendo uma “peça” de software constituído por “peças” menores, que fornecem inúmeras funcionalidades que podem ser re-utilizadas. Relembra-se a título de exemplo que, no âmbito deste projecto, se usou as *frameworks* Acegi e JAAS às quais foi delegada a autenticação. O uso de *proxy web services* também evitará a implementação de uma classe que apenas suporte os parâmetros do serviço de entrada.

Facilidade de manutenção que advém directamente do ponto anterior. Uma vez que não se tem código, a manutenção fica desde logo facilitada.

Facilidade de configuração pretende-se resolver o problema dos mediadores desenvolvendo uma aplicação apenas baseada na configuração das *frameworks* já existentes e distribuídas juntamente com a *framework* Mule ESB.

Flexibilidade que já foi apresentada no Capítulo 4 como sendo uma das vantagens desta tecnologia é a facilidade com que se acrescentam aplicações na arquitectura de integração.

Uniformizar as aplicações da empresa que está a desenvolver outros projectos recorrendo ao Mule, o que conduz a um melhor entendimento da tecnologia e consequentemente facilita a sua manutenção e evolução.

Reutilização de código que é consequência directa do que foi mencionado no *item* acima. Dado que existe alguma uniformização nas aplicações da empresa a reutilização de código é favorecida. Imagine-se que se melhora um determinado aspecto do componente de *logging*, como por exemplo, tornar os *logs* mais amigáveis. Este componente poderá desde logo ser reutilizado no *bus* dos mediadores.

Note-se que é possível estabelecer uma analogia entre um mediador de seguros e um *Enterprise Service Bus*. Esta analogia, não terá nenhum interesse científico, mas não deixa de ser caricata, pelo que se julga relevante apresentá-la.

Um mediador de seguros é um intermediário entre o tomador do seguro e a seguradora, sendo que o tomador pode dirigir-se ao mediador para efectuar o pagamento do prémio ou expor dúvidas que possa ter em relação à(s) sua(s) apólice(s). Um tomador comum, poderá não conhecer os conceitos e vocabulário inerentes ao negócio dos seguros, pelo que passa a expor um problema ou dúvida ao mediador que se encarrega de transmitir o problema, ou dúvida, à seguradora, usando possivelmente termos mais técnicos de modo a que seja facilitada a compreensão do problema do tomador junto da companhia. Um *Enterprise Service Bus* desempenha exactamente a mesma função de mediação, sendo recorrente na literatura, apelidar os *Enterprise Service Bus* de *brokers*. De facto, e como já foi várias vezes referido neste documento, os ESB's recebem uma mensagem de uma determinada aplicação ligada ao *bus* e assumem a responsabilidade de a transformar, se for necessário, para que posteriormente esta possa ser encaminhada e entendida pela aplicação de destino.

5.2 Tecnologias Envolvidas

Esta secção pretende dar ao leitor uma visão global do leque de tecnologias envolvidas nesta solução. Atendendo a uma das definições que se deu aos *Enterprise Service Bus* no Capítulo 4, afirma-se, que estes e mais particularmente a *framework* Mule ESB podem ser vistos como uma *framework* que abarca um conjunto de outras *frameworks*. Assim sendo, quando se opta pela utilização de semelhante tecnologia para solucionar problemas de integração de *software*, a implicação que resulta é um conhecimento razoável das tecnologias abarcadas pelo *bus*, uma vez que se pretende atingir uma solução, com a menor quantidade possível de código próprio.

Segue uma breve apresentação do conjunto de tecnologias envolvidas na solução do problema de integração de mediadores com companhias de seguros.

5.2.1 Spring Framework

Esta *framework* é desenvolvida em Java e muito usada no desenvolvimento de aplicações seguindo o padrão MVC¹. Spring apoia-se principalmente em três conceitos [Spr05]:

Injecção de dependências — Padrão IoC (*Inversion of Control*) que define o princípio que o *workflow* de execução não é da responsabilidade da aplicação propriamente dita, mas da *framework* associada.

¹ Padrão de separação da aplicação em camadas: *Model-View-Controller*.

Programação orientada a aspectos permite separar as considerações técnicas (*aspects*) das descrições do negócio, numa determinada aplicação.

Camada de abstracção permite a fácil integração com outras *frameworks*.

Esta tecnologia é usada no Mule para configuração do *bus* nomeadamente, dos seus componentes.

5.2.2 Axis

Esta *framework* desenvolvida em Java, insere-se na categoria dos motores de *web services* (*web services engine*). É um projecto desenvolvido pela Apache Foundation² que fornece [Fou05a]:

Uma API cujo objectivo é o desenvolvimento de *web services* SOAP RPC, ou SOAP orientado a mensagens.

Suporte para vários transportes nomeadamente: HTTP, SMTP, FTP, POP, IMAP ...

Serialização/Desserialização de objectos Java em mensagens SOAP.

Ferramentas para a criação de WSDL com base em classes Java e inversamente a criação de classes Java com base num WSDL (classes *proxies*) para invocação de *web services*.

Ferramentas para a distribuição (*deployment*), teste e monitorização de *web services*.

Esta *framework* foi usada inicialmente, aquando da familiarização com o Mule, para expor e invocar *web services*. Devido às dificuldades encontradas para configuração da segurança no Mule e configuração de *proxies* de *web services*, decidiu-se usar o motor CXF.

5.2.3 CXF

Esta *framework* é igualmente um motor de *web services*, também desenvolvida pela Apache Foundation. É a fusão de dois projectos anteriores: Xfire e Celtix. Dispõe, além das características apresentadas para a *framework* Axis, das seguintes funcionalidades (entre outras) [Fou05b] :

Suporte de Standards incluindo: WS-Addressing, WS-Policy, WS-ReliableMessaging, and WS-Security.

Suporte para os seguintes transportes: CORBA, RESTful HTTP.

²Página oficial disponível em: <http://www.apache.org/>

Possibilidade de conjugação com a *framework* Spring.

Suporte de *web services proxies*.

5.2.4 WSS4J

Mais uma vez se fez uso de uma *framework* da Apache Foundation. WSS4J foi usada neste projecto para a configuração da segurança, mais precisamente da autenticação dos pedidos feitos pela aplicação Gis Mediadores ao servidor Mule e consiste numa implementação para Java da especificação WS-Security do grupo OASIS, mais precisamente:

- Web Services Security: SOAP Message Security 1.1;
- Username Token Profile 1.1;
- X.509 Certificate Token Profile 1.1

Esta *framework* permite interoperabilidade com servidores/clientes .NET [Fou09] e dispõe das seguintes funcionalidades:

- XML Security
 - XML Signature
 - XML Encryption
- Tokens
 - Username Tokens
 - Timestamps
 - SAML Tokens

5.2.5 Acegi

A esta tecnologia, baseada na *framework* Spring, foi delegada a tarefa de validação da autenticação. Após um determinado *web service* ter recebido as credenciais do cliente, é necessário definir a fonte de autenticação (base de dados relacional, *Active Directory*, Sistema Operativo etc.), sendo apenas necessário definir as características da fonte (*Authentication Provider*).

5.2.6 JAAS

JAAS é mais uma *framework* de segurança para aplicações desenvolvidas em Java. É integrada no JRE a partir da versão 1.4, permite autenticar utilizadores e atribuir-lhes permissões[[Sun01](#)]. Esta tecnologia foi usada em conjunto com a *framework* Acegi, para autenticação junto do sistema operativo, isto é, valida um pedido, se as credenciais representarem um utilizador do sistema operativo. Esta *framework* foi usada da seguinte forma:

1. Num primeiro tempo delega-se a autenticação à *framework* Acegi;
2. Esta última, por sua vez, delega a autenticação a JAAS para autenticação do sistema operativo, já que Acegi não dispõe desta funcionalidade de autenticação.

5.2.7 XSLT

Esta tecnologia é essencial, tanto para a viabilidade da utilização *Enterprise Service Bus* para o problema de integração dos mediadores, como para qualquer problema que tenha de ser solucionado por um ESB. É uma tecnologia chave, dado que, actualmente, grande parte da informação trocada entre aplicações é feita no formato XML. A habilidade de se poder transformar essas mensagens é primordial. XSLT é definida no seio da especificação XSL do W3C e permite a transformação de uma determinada mensagem XML, em outra mensagem XML ou HTML, por exemplo. Uma transformação XSLT tem obrigatoriamente de se apoiar num documento no formato XML. As transformações são baseadas em interrogações ao documento original recorrendo à linguagem de interrogação XPath [[Cla99](#)].

5.3 Comparação de Arquitecturas

Foram avaliadas neste documento, duas tecnologias capazes de solucionar o problema de integração entre mediadores e companhias de seguros. *Web services* e ESB's foram estudados separadamente. Resta então, proceder-se a uma comparação das tecnologias e ver como estas satisfazem os aspectos desejáveis para uma boa integração, enunciados no Capítulo 2.

5.3.1 Tecnologias

No Capítulo 3 afirmou-se que, apesar da densa especificação dos *web services* SOAP, a implementação destes é relativamente simples, graças nomeadamente a *frameworks* tais como, Axis ou CXF, apresentadas na secção 5.2. O programador não precisa de saber

escrever uma WSDL ou construir uma mensagem SOAP, uma vez que já existem ferramentas que tratam (e bem) desse aspecto. O desenvolvimento de *web services* resume-se então à utilização dessas ferramentas e à implementação da lógica de negócio que se pretende expor. As tecnologias envolvidas nas ligações ponto-a-ponto, resumem-se essencialmente à linguagem de programação que se adopta. Obviamente que é necessário conhecer os conceitos abarcados pelos *web services*, mas a utilização desta tecnologia não requer conhecimentos profundos acerca de SOAP e WSDL.

Um *Enterprise Service Bus* é constituído por várias *frameworks* de terceiras partes, que oferecem inúmeras funcionalidades. O Mule faz uso de certas características da *framework* Spring, nomeadamente para a configuração, pelo que é necessário ter conhecimentos básicos dessa tecnologia. Lembre-se a prova de conceito “*Web Services Proxy*” em que se pretende expor um *web service* de “fachada”, uma espécie de *gateway* sem ter que desenvolver uma classe cuja função seria apenas suportar os dados de entrada. O desenvolvimento desta prova de conceito implica um conhecimento mais profundo da tecnologia WSDL, uma vez que a WSDL tem de ser construída manualmente sem recorrer a ferramentas.

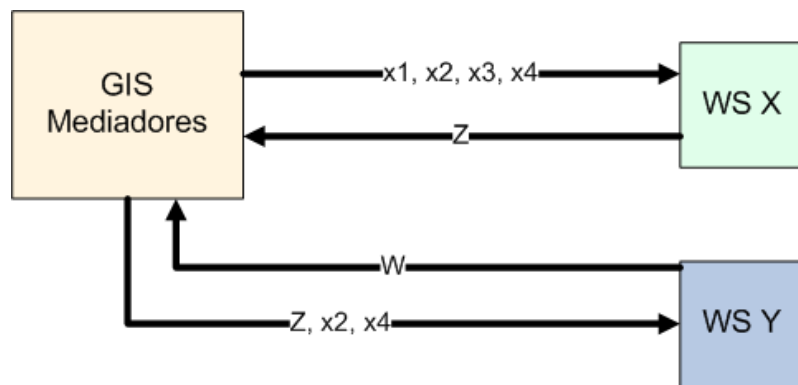
5.3.2 Segurança

Em termos de segurança, as duas tecnologias equivalem-se, sendo que, as duas abordagens fazem uso da *framework* WSS4J, apresentada na Secção 5.2. No entanto, é possível delegar a validação da autenticação a outras *frameworks*, quando se opta pela utilização de um ESB como o Mule, enquanto que no caso das ligações ponto-a-ponto, sendo o motor de *web services* o Axis, terá de ser programada uma rotina de *callback* para efeitos de validação de autenticação.

5.3.3 Flexibilidade

A flexibilidade representa a abertura de uma aplicação à introdução de alterações. Uma das formas de a conseguir, consiste em exteriorizar, certos aspectos configuráveis de uma aplicação, como por exemplo ligações a bases de dados. Exteriorizar, significa que estas configurações são declaradas em ficheiros XML (apenas uma vez), em vez de estarem codificadas em duro na aplicação. Um dos incentivos na utilização da *framework*, é precisamente conseguir desenvolver uma aplicação altamente configurável, evitando a escrita de código.

Admitindo-se a adopção de ligações ponto-a-ponto, num cenário favorável, a informação pretendida pelo mediador é apenas o retorno directo de um determinado *web service*. Num cenário mais complexo, a informação pretendida resulta da invocação em cadeia de vários serviços. A Figura 5.2 ilustra precisamente um desses cenários em que, para obter

Figura 5.2: Exemplo de Invocação Complexa de *Web Services*

uma determinada informação w , invoca-se primeiramente um serviço x com os parâmetros $x1, x2, x3, x4$, que devolve o resultado z , ao qual foi necessário aplicar um conjunto de métodos para transformar o seu formato. De seguida invoca-se um serviço y com o resultado z (já transformado) e mais dois dos parâmetros com os quais foi chamado o serviço x . Toda esta cadeia de invocações e o processamento resultante fica a cargo da aplicação dos mediadores, sobrecarrega a aplicação e pode alterar a sua estabilidade. O Mule resolve essas questões, uma vez que se apresenta com a capacidade de transformar, filtrar e encaminhar de forma inteligente mensagens, entre aplicações ligadas ao *bus*.

5.3.4 Escalabilidade

A escalabilidade prende-se com a capacidade de acréscimo de aplicações à arquitectura de integração, sem originar alterações significativas. O Mule tem essa capacidade, sendo até, uma das características principais de tais tecnologias. A ligação de uma aplicação ao *bus*, abre desde logo a possibilidade de comunicação com as outras aplicações igualmente ligadas ao *bus*. Cada aplicação apenas se preocupa em comunicar com o ESB e desconhece totalmente as outras aplicações, o que permite que uma determinada aplicação ligada ao ESB, possa ser livremente alterada. As restantes não sofrem qualquer alteração, nem sequer sabem das alterações.

As ligações ponto-a-ponto, quanto a si, são dificilmente escaláveis, sendo que a adição de uma nova aplicação à arquitectura existente pode originar alterações consideráveis. É certo que actualmente a necessidade de integração dos mediadores com companhias de seguros envolve apenas duas aplicações (a dos mediadores e a das companhias) e para tais arquitecturas as ligações ponto-a-ponto adequam-se perfeitamente. No entanto, não é possível prever como evoluirá o negócio e talvez, algures no futuro seja necessário acrescentar outras aplicações à arquitectura de integração. A utilização do Mule é uma forma de se precaver para as futuras necessidades de integração.

5.3.5 Manutenção

A empresa I2S está organizada em sectores, dos quais são destacados os dois que interessam para este projecto: sector dos mediadores e o sector CID³ (Concepção, Investigação e Desenvolvimento). A arquitectura de integração para o problema abordado neste relatório é concebida e será implementada no sector CID, sendo que posteriormente a manutenção ficará a cargo do sector dos mediadores.

Como já foi referido a manutenção de *software* ocupa um lugar preponderante no ciclo de vida de uma aplicação. A dificuldade de manutenção está estreitamente ligada com a quantidade de código escrito na fase de implementação. Manter código é uma tarefa árdua, pelo que um conhecimento profundo do código intrínseco a uma aplicação é fundamental para uma boa manutenção. Na subsecção 5.3.3 já se concluiu que a arquitectura que promove as ligações ponto-a-ponto implica a escrita de código⁴, nomeadamente, no caso de se ter invocações complexas, à semelhança da invocação da Figura 5.2, enquanto que o Mule pretende atingir a integração sem a escrita de uma linha de código, recorrendo apenas à linguagem XML para configuração.

É legítimo, neste momento o leitor contradizer o autor, pensando que a manutenção do XML de configuração, também deve ser considerada manutenção de código. É óbvio que sim! No entanto, existe um *plug-in* gráfico, apresentado na subsecção 4.3.1.2 que permite a configuração do circuito da mensagem ao longo do *bus* de integração. O uso deste *plug-in* deve permitir desenhar graficamente o *workflow* dentro do bus, ocultando o código XML inerente ao mesmo. Infelizmente, durante este projecto, testemunhou-se que o IDE está bastante aquém do que pretende ser. A MuleSource, organização criadora da *framework* Mule, descontinuou o desenvolvimento do *plug-in* por o julgar demasiado ambicioso[Olm09].

5.3.6 Sistemas Legados

Neste projecto, existem sistemas legados tanto do lado dos mediadores, como do lado das companhias. Os sistemas legados constituem um dilema para as empresas, como já foi referido na Secção 2.3.5. Por um lado, são fruto de anos de trabalho, abarcando um volume de informação considerável e suportando o negócio, há mais de uma década, no caso concreto dos mediadores. Por outro lado, estes sistemas são desenvolvidos em linguagens de programação hoje em dia obsoletas, dificultando desta forma a integração com tecnologias mais recentes.

Os módulos de transferência de dados do Gis Mediadores começaram a ser desenvolvidos a partir de 1992 e são usados, nomeadamente, para transferência de ficheiros. São módulos que são usados com alguma frequência pelos mediadores e são necessários para

³É neste sector que o autor deste documento desenvolveu este projecto

⁴Entenda-se do lado do cliente

suportar o negócio destes. A grande maioria destes módulos foram desenvolvidos em C++, recorrendo-se a DCOM para a comunicação com a aplicação Gis Mediadores.

Actualmente, a necessidade de integração necessita apenas da ligação da aplicação Gis Mediadores com a aplicação das companhias, mais precisamente com os *web services* por elas expostos, sendo que a ligação entre a aplicação principal dos mediadores com os seus módulos mantém-se inalterada. De futuro, se os módulos necessitarem comunicar directamente com as seguradoras e encarando uma arquitectura de integração ponto-a-ponto, cada um dos módulos terá de estabelecer uma ligação com cada serviço da companhia e encarregar-se do processamento, filtragem e transformação de mensagens exigidos pela integração. Recorrendo-se a um *bus*, os módulos apenas necessitam de estabelecer uma ligação ao ESB e ficam desde logo preparados para comunicar com todas as aplicações envolvidas na arquitectura de integração.

Alguns dos serviços pretendidos pelos mediadores já existem do lado da companhia, sendo estes apenas usados internamente. Cada companhia dispõe de um *Service Manager* e de um *Service Provider*. O *Service Manager* dispõe de uma lista de serviços e mantém informação acerca deles, nomeadamente se estão “em pé” ou parados. Quando a aplicação da companhia solicita um determinado serviço ao *Service Provider*, este verifica junto do *Service Manager* se o serviço está “em pé”. Se efectivamente estiver, o *Service Provider* devolve a ligação para o serviço em questão à aplicação que o requisitou, caso contrário o serviço é arrancado e seguidamente, é devolvida a ligação para esse serviço. Estes componentes foram implementados em linguagem Java, sendo a comunicação entre o *Service Manager* e *Service Provider* possível, graças à tecnologia Java RMI. Obviamente que não é viável para as companhias expor os serviços “para fora” da mesma forma como são expostos internamente. Como já foi referido no Capítulo 3, estas tecnologias de distribuição apresentam deficiências, principalmente quando a rede de comunicação é a *Internet* e que se exige segurança. Relembra-se que o Java RMI apenas permite comunicação entre aplicações desenvolvidas em Java e que o transporte usado é TCP, pelo que não é possível implementar segurança em cima deste protocolo. Além de mais, a exposição desses serviços implica abrir uma porta, o que causa obviamente problemas de segurança para a I2S. Note-se que *Service Provider* e *Service Manager* desempenham uma função semelhante à do serviço de directório para *web services*, UDDI, analisado no Capítulo 3.

Ao longo deste documento tem-se referenciado os *web services* das companhias. Se a funcionalidade pretendida pelos mediadores já existir para propósitos internos então, o *web service* exposto, consiste apenas num *proxy*, para um serviço do *Service Provider*. Assim sendo, o *web service* exposto consiste apenas, numa camada intermédia de acesso à funcionalidade já existente internamente. Quando o serviço pretendido não existe internamente, então deverá ser avaliada a utilidade do serviço para uso interno da companhia. Se se julgar relevante, o *web service* será desenvolvido usando a estratégia anterior, caso contrário, seria encapsulada a lógica de negócio do serviço num *web service* tradicional.

5.3.7 Custo Inicial

Já foram comparados todos os aspectos enunciados no Capítulo 2. Existe no entanto, outro aspecto digno de relevo — o custo inicial destas soluções. Dum ponto de vista financeiro, não existem custos, uma vez que se faz uso de tecnologias livres e obviamente que a empresa já possui licenças para o uso de tecnologias como o Visual Studio 2005, um IDE que permite o desenvolvimento de aplicações .NET. O custo inicial que se refere nesta secção, faz referência essencialmente ao tempo despendido na implementação da arquitectura de integração.

No caso de se usar o Mule, o custo inicial é maior, uma vez que envolve, num primeiro tempo o estudo da tecnologia, uma familiarização à mesma e num segundo tempo, uma análise de viabilidade à adequação da tecnologia ao problema de integração. Só depois de se completar estas duas fases é que se pode proceder à implementação da solução propriamente dita. Estima-se no entanto, que este custo inicial mais elevado, seja no futuro compensado por uma mais fácil manutenção do que as ligações ponto-a-ponto.

5.4 Conclusões

Já foram discutidas neste documento as duas alternativas encaradas como solução para o problema de integração entre mediadores e companhias de seguros, os *web services* no Capítulo 3 e os *Enterprise Service Bus* no Capítulo 4. Este capítulo comparou as duas arquitecturas seguindo os aspectos a considerar, enunciados no Capítulo 2. É agora tempo, de se tomar uma decisão.

Ao longo deste capítulo foi evidenciado que ambas as soluções são viáveis, sendo que ambas apresentam vantagens e desvantagens, em relação uma à outra. As ligações ponto-a-ponto providenciadas pelos *web services* não envolvem um estudo considerável, consistem apenas no desenvolvimento de um conjunto de *proxies* do lado dos mediadores, usando para tal a mesma linguagem de programação que a aplicação Gis Mediadores, bem conhecida por parte destes. Pelo contrário, o Mule é ainda uma tecnologia recente e mais complexa do que a anterior, sendo necessário atribuir um tempo maior para uma análise mais profunda. Mule é baseado em Java, uma tecnologia com a qual os mediadores não têm um elevado *know-how*, o que constitui um ponto de renitência para estes, que receiam uma manutenção dificultada. Os *web services* adequam-se perfeitamente à resolução do problema, sendo estes capazes de veicular informação entre os parceiros de negócio de forma segura. Apresentam uma certa flexibilidade e escalabilidade, mas não igualável à dos *Enterprise Service Bus*. No que diz respeito a sistemas legados, ambas as tecnologias permitem a sua inclusão na arquitectura de integração. Prevê-se uma manutenção facilitada, decorrente da utilização de um ESB, uma vez que não prevê a escrita de código, prevê sim, a implementação de uma solução altamente configurável. A escolha

não resulta numa tarefa fácil, uma vez que nenhuma tecnologia se afirma claramente em relação à outra.

Precaver-se para o futuro foi o critério de desempate. Este critério é essencial, sob pena de se incorrer em prejuízos maiores, isto é, uma solução ajustada ao problema actual, poderá não conseguir atender os desafios do futuro, pelo que eventualmente se terá de voltar atrás para remediar as deficiências das soluções do passado. Nesta perspectiva, o Mule destaca-se claramente em relação aos *web services* e atendendo ao facto de já se ter gasto tempo na análise dos *Enterprise Service Bus*, estes serão apresentados aos mediadores como sendo a solução mais adequada ao problema de integração com as companhias de seguros. Note-se finalmente que caberá exclusivamente a estes validar a solução proposta.

Capítulo 6

Conclusões

Este capítulo final serve de conclusão para este trabalho. Primeiramente, faz-se uma síntese do trabalho efectuado ao longo deste projecto, apreciando a satisfação dos objetivos. De seguida, são feitas as considerações finais acerca da integração de *software* e, finalmente, perspectivam-se os trabalhos que decorrerão futuramente.

6.1 Satisfação dos Objectivos

Os mediadores, clientes da I2S possuem uma aplicação de gestão relativa ao seu negócio. Mediadores e companhias partilham a mesma informação, nomeadamente informação relativa a clientes, apólices etc.. Assim sendo, mediadores e companhias têm a necessidade de ter essa informação sincronizada, pelo que os mediadores devem poder solicitar serviços que devolvam a informação pretendida, em tempo real.

Actualmente, a informação é obtida de forma diferida, através do envio periódico de ficheiros ou por troca de *e-mails*. A necessidade de um maior grau de integração é evidente, pelo que este projecto assentou na avaliação de duas possíveis arquitecturas de integração, como solução para o problema enunciado: Ligações ponto-a-ponto, recorrendo-se a *web services*, ou arquitectura em *bus*, em que este age como intermediário na comunicação, sendo ele responsável pelo encaminhamento dos pedidos.

As ligações ponto-a-ponto foram estudadas e postas em prática por meio da implementação de um *web service* para empresa de peritagem VIAPRE¹, que pretende disponibilizar à seguradora AXA² a possibilidade de requerer uma peritagem de forma automática. As análises feitas às ligações ponto-a-ponto evidenciaram uma arquitectura fácil de

¹Página oficial disponível em: <http://www.viapre.pt>.

²Página oficial disponível em : <http://www.axa.pt>.

implementar, no entanto, quando o número das aplicações da arquitectura de integração tendem a aumentar, a gestão dessas ligações torna-se difícil.

Os objectivos para este projecto foram atingidos, na medida em que se pode neste momento concluir sobre a arquitectura mais adequada para a resolução do problema — *Enterprise Service Bus*. O estudo desta tecnologia, através da realização de provas de conceito, revelou que é possível atender este problema, implementando uma solução altamente configurável e sem necessidade de escrita de código Java (ou noutra linguagem de programação). A flexibilidade, escalabilidade e facilidade de manutenção que esta solução apresenta, compensam largamente o custo inicial, que é maior do que em ligações ponto-ponto.

As ligações ponto-a-ponto são uma solução perfeitamente aceitável, sendo o custo inicial necessário mais reduzido do que o dos *Enterprise Service Bus* assim como a sua implementação mais trivial. No entanto, solucionam apenas o problema actual sendo que, tal arquitectura, não garante o atendimento de problemas futuros.

6.2 Considerações Finais

A globalização dos mercados, o elevado grau de informatização dos mesmos e a necessidade de partilha de informação entre os vários parceiros de negócio, conduz a uma necessidade de integração³ inevitável, sendo este, actualmente, um dos problemas centrais da Engenharia de *Software*.

O extremo dinamismo dos mercados, obriga a uma evolução tecnológica em paralelo, isto é, a tecnologia tem de evoluir paralelamente aos mercados de modo a poder satisfazer as exigências destes. As primeiras tecnologias de distribuição permitiram a execução de funcionalidade remotamente, em ambientes controlados como, por exemplo, redes locais. A expansão da *Internet*, ofereceu um meio para integração de *software* entre parceiros de negócio. As limitações das tecnologias de distribuição em termos de interoperabilidade, proporcionaram o nascimento dos *web services* que, devido à utilização de *standards*, garantiram essa interoperabilidade. As vantagens competitivas e a agilização dos processos de negócio que advêm de uma visão integrada dos negócios dos parceiros estratégicos, criaram a tentação de adicionar cada vez mais aplicações à arquitectura de integração, o que levou a que, rapidamente, as ligações pontuais proporcionadas pelos *web services*, se tornassem difíceis de gerir. A evolução lógica da tecnologia conduziu à criação dos *Enterprise Service Bus* cuja arquitectura em *bus*, oferece flexibilidade e escalabilidade a futuros desafios.

O processo de integração de *software*, é um processo complexo. Não existe nenhuma metodologia sistemática que permita solucionar qualquer problema de integração. Para cada caso, é necessário estabelecer os requisitos e consoante esses requisitos, definir as

³Do ponto de vista informático.

características desejáveis para se atingir um bom nível de integração. Precaver-se para o futuro é fundamental, quando se aborda problemas de integração. Não se deve apenas solucionar um problema actual, deve manter-se uma porta em aberto para o futuro. É extremamente difícil prever como evoluirá o negócio, ou quais serão os desafios futuros, no entanto é certo que esses desafios vão surgir.

As tecnologias de integração referidas neste projecto de mestrado, apresentaram sempre um futuro promissor aquando da sua criação. No entanto, as mudanças mais ou menos repentinas e mais ou menos drásticas dos mercados, acabaram por evidenciar as limitações destas tecnologias. Neste momento, os *Enterprise Service Bus* são a nova vaga de Aplicações de Integração Empresarial (*Enterprise Integration Application*), sendo-lhes reservado um futuro promissor. Até quando? Serão realmente, os *Enterprise Services Bus* capazes de responder aos futuros desafios da integração de *software*?

6.3 Perspectivas de Trabalhos Futuros

A prova de conceito — *web service proxy*, não foi concluída devido às limitações da versão⁴ da *framework* utilizada. Após a conclusão desta dissertação, será terminada esta prova de conceito e preparada a apresentação da solução proposta aos mediadores. Caberá exclusivamente a estes validar os *Enterprise Service Bus* como arquitectura de integração, para o caso de mediadores e companhias de seguros.

Se esta solução for validada, dar-se-á início à sua implementação, no entanto, note-se que seria interessante usar o *plug-in* gráfico da *framework* Mule, como suporte à configuração do *workflow* dentro do *bus*, resultando numa manutenção facilitada.

Se a solução proposta não for suficientemente convincente para os mediadores decidirem adoptá-la para resolver o seu problema, será dado início ao desenvolvimento das ligações ponto-a-ponto, inicialmente sugeridas pelos mediadores.

⁴versão 2.0.2

Referências

- [Afo09] Manuel Afonso. Comunicação de dados entre as seguradoras e gis mediadores, 2009. Documento Interno.
- [AG02] Arnaud Sahuguet Aniruddha Gokhale, Bharat Kumar. Reinventing the wheel? corba vs. web services. *Desconhecido*, Acedida pela última vez em 2002. <http://www2002.org/CDROM/alternate/395/>.
- [Ale04] Ben Alex. *Acegi Security System for Spring*. Acegi, Acedida pela última vez em 2004. Disponível em <http://www.acegisecurity.org/>.
- [Bal05] Naveen Balani. Model and build esb soa frameworks. *IBM DeveloperWorks*, Acedida pela última vez em 2005. <http://www.ibm.com/developerworks/web/library/wa-soaesp/>.
- [Bar] Douglas K. Barry. Service-oriented architecture (soa). <http://www.service-architecture.com/contact.html>.
- [Bes] BestPriceComputers. Enterprise integration application. <http://www.bestpricecomputers.co.uk/glossary/enterprise-application-integration.htm>.
- [Boo04] David Booth. Web services architecture. *W3C Working Group Note*, Acedida pela última vez em 2004. <http://www.w3.org/TR/ws-arch/>.
- [Bor03] Xavier Borderie. L'architecture rest. *Journal du Net*, Acedida pela última vez em 2003. http://www.journaldunet.com/developpeur/tutoriel/xml/030707xml_rest1a.shtml.
- [Bor04] Xavier Borderie. Soap, xml-rpc et rest : différences et intérêts. *Journal du Net*, Acedida pela última vez em 2004. <http://en.wikipedia.org/wiki/XML-RPC/>.
- [Bus03] Christoph Bussler. *B2B Integration Concepts and Architecture*, chapter 1.1. Springer, 2003.
- [Cla99] James Clark. Web services architecture. *W3C Recommendation*, 1999. Disponível em <http://www.w3.org/TR/xslt>.
- [Far07] Nicolas Farges. Enterprise service bus (esb): tendance de fond ou nouveau "buzzword"? *ZdNet*, Acedida pela última vez em 2007. <http://www.zdnet.fr/actualites/informatique/0,39040745,2132625,00.htm>.

REFERÊNCIAS

- [Fie00] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, IRVINE, Acedida pela última vez em 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [Fou05a] Apache Foundation. *Axis User's Guide*. Apache Foudation, Acedida pela última vez em 2005. Disponível em <http://ws.apache.org/axis/java/user-guide.html>.
- [Fou05b] Apache Foundation. *CXF User's Guide*. Apache Foudation, Acedida pela última vez em 2005. Disponível em <http://cxf.apache.org/>.
- [Fou09] Apache Foundation. Apache wss4j, Acedida pela última vez em 2009. http://ws.apache.org/wss4j/#What_is_WSS4J.
- [GH05a] Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns*, chapter 1.6. Addison-Wesley, 2005.
- [GH05b] Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns*, chapter 1.2. Addison-Wesley, 2005.
- [GH05c] Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns*, chapter 2.6. Addison-Wesley, 2005.
- [GH05d] Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns*, chapter 2.7. Addison-Wesley, 2005.
- [Gis01] Dan Gisolfi. Is web services the reincarnation of corba? *Developer Works (IBM)*, Acedida pela última vez em 2001. <http://www.ibm.com/developerworks/webservices/library/ws-arc3/>.
- [Lar07] David Lardet. Pour ne plus être en rest, comprendre cette architecture. *Bio-loGeek*, Acedida pela última vez em 2007. http://www.journaldunet.com/developpeur/tutoriel/xml/030707xml_rest1a.shtml.
- [McG05] Jim McGee. Legacy systems: Why history matters. *Enterprise Systems*, Acedida pela última vez em 2005. <http://esj.com/news/article.aspx?EditorialsID=1529>.
- [Mul08] MuleSource. *Mule User Guide*. MuleSource, Acedida pela última vez em 2008. Disponível em <http://www.mulesource.org/display/MULE2USER/Proxying+Web+Services>.
- [Nad06] Anthony Nadalin. Oasis web services security (wss) tc. *OASIS*, Acedida pela última vez em 2006. Disponível em http://www.oasis-open.org/committees/documents.php?wg_abbrev=wss.
- [Nic04] Matthieu Nicolescu. *WebServices*. SUPINFO, 2004.
- [Olm09] Dirk Olmes. *Mule IDE Manual*. MuleSource, Acedida pela última vez em 2009. Disponível em <http://www.mulesource.org/display/MULEIDE/Home>.

REFERÊNCIAS

- [Pin03] Sílvio Gomes Pinheiro. Eai - enterprise application integration. Technical report, Instituto Superior de Engenharia do Porto, ISEP, 2003.
- [Pre97] Roger S. Pressman. *Software Engineering - A practitioner's approach*, chapter 27. Mac Grow Hill, 4^a edition, 1997.
- [PT04] ITtoolbox Popular e Team. Difference between soa and esb. *ITtoolbox*, Acedida pela última vez em 2004. <http://eai.ittoolbox.com/documents/difference-between-soa-and-esb-16734>.
- [Sne01] James Snell. *Programming Web Services with SOAP*, chapter 1.2. O'Reilly, 1^a edition, 2001.
- [Som00] Ian Sommerville. *Software Engineering*, chapter 26. Addison-Wesley, 6^a edition, 2000.
- [Spr05] Spring. Spring framework, Acedida pela última vez em 2005. Disponível em: <http://www.springsource.org/>.
- [Sun01] Sun. *JAAS Reference Manual*. Sun, Acedida pela última vez em 2001. Disponível em <http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html>.
- [W3S] W3Schools. Soap tutorial. <http://www.w3schools.com/soap/default.asp>.
- [WDB01] Douglas Marks William D. Brosey, Richard Neal. Grand challenges of enterprise integration. Technical report, BWXT Y-12 LLC. - IMTI Inc. - Pinnacle Communication Services, 2001.
- [Whi75] James E. White. A high-level framework for network-based resource sharing. *IEEE*, Acedida pela última vez em 1975. <http://tools.ietf.org/html/rfc707>.
- [Wik09a] Wikipedia. Assurance. *Wikipedia*, Acedida pela última vez em 2009. Disponível em : <http://fr.wikipedia.org/wiki/Assurance>.
- [Wik09b] Wikipedia. Enterprise application integration. *Wikipedia*, Acedida pela última vez em 2009. http://en.wikipedia.org/wiki/Enterprise_application_integrationIntegration_patterns/.
- [Wik09c] Wikipedia. Enterprise service bus. *Wikipedia*, Acedida pela última vez em 2009. http://fr.wikipedia.org/wiki/Enterprise_Service_Bus.
- [Wik09d] Wikipedia. Xml-rpc. *Wikipedia*, Acedida pela última vez em 2009. <http://en.wikipedia.org/wiki/XML-RPC/>.
- [Wik09e] Wikipedia. Message-oriented middleware. *Wikipedia*, Acedida pela última vez em 2009. http://en.wikipedia.org/wiki/Message-oriented_middleware.

REFERÊNCIAS

- [WRC06] Janel Barfield William R. Cook. Web services versus distributed objects: A case study of performance and interface design. *University of Texas*, 2006. Disponível em <http://www.cs.utexas.edu/~wcook/Drafts/2006/WSvsDO.pdf>.
- [Zou02] Federico Zoufaly. Issues and challenges facing legacy systems. *Developer.com*, Acedida pela última vez em 2002. http://www.developer.com/mgmt/article.php/11085_1492531_1.