Faculdade de Engenharia da Universidade do Porto

Departamento de Engenharia Electrotécnica e de Computadores

Criação Rápida de Serviços: Tradução de SDL para ANSA

Paula Cristina Ribeiro Coutinho de Oliveira

Licenciada em Engenharia Electrotécnica, Ramo de Electrónica, Instrumentação e Computação pela Universidade de Trás-os-Montes e Alto Douro

Dissertação submetida para satisfação parcial dos requisitos do grau de Mestre em Engenharia Electrotécnica e de Computadores (Área de especialização de Telecomunicações)

Porto, Setembro de 1995

Dissertação realizada sob a supervisão do

Prof. Doutor Eurico Manuel Elias de Morais Carrapatoso

Professor Auxiliar do
Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

À Cândida
À Xanda e ao Jorge

Resumo

Actualmente, a tendência de evolução das redes de comunicação é no sentido de poderem proporcionar aos utilizadores uma maior variedade de serviços capazes de satisfazer necessidades muito diversas. Estes serviços são em muitos casos novos e de criação complexa dadas as funcionalidades oferecidas e a distribuição entre vários componentes. A necessidade de metodologias e ferramentas que facilite o processo de desenvolvimento é pois imperiosa.

O SDL (*Specification Description Language*) é uma linguagem que permite descrever e especificar serviços de uma forma rápida, sendo no entanto necessária, em muitos casos, a sua posterior distribuição por diferentes máquinas. Essas diferenças tanto dizem respeito ao hardware como ao Sistema Operativo utilizado.

O GEODE é uma ferramenta que oferece um ambiente SDL, tendo sido criado para suportar o ciclo de desenvolvimento desde o projecto à geração de código e de testes. Para isso, integra um conjunto de ferramentas para a edição, a verificação sintáctica e semântica, a simulação e a geração de código para uma aplicação correr num dado sistema.

A distribuição de serviços pode ser conseguida através de plataformas distribuídas, das quais o ANSA é um exemplo. Esta plataforma suporta o desenvolvimento e construção de aplicações distribuídas independentemente das máquinas em que vão correr.

Com base nestas duas ferramentas pretendeu-se tornar possível uma criação rápida de serviços, traduzindo os ficheiros gerados pelo GEODE, aquando da criação de um serviço, em ficheiros suportados pelo ANSA para posterior distribuição.

Neste contexto, é feita nesta tese uma apresentação sobre serviços, mais especificamente, serviços multimédia e uma descrição das ferramentas e facilidades dos dois pacotes de software, referidos anteriormente. É ainda proposta uma implementação para a tradução pretendida e descrita uma aplicação.

Palavras Chave

Multimédia	Bases de Dados	Sistemas Distribuídos
Especificação Formal	Orientado a Objectos	

Abstract

Communications Networks are in continuos progress with the aim of providing a wider variety of services to satisfy the demand of a number of very different users.

SDL - Specification Description Language is a language which allows a quick description and specification of services, being nevertheless necessary in many cases its subsequent distribution through different machines. Those differences refer both to hardware and Operating Systems.

GEODE is a tool that offers an SDL environment to support the development cycle from the project to the code generation and tests. To this effect, it integrates a set of tools for edition, syntactic and semantic verification, simulation and code generation.

The distribution of services may be achieved through distributed platforms of which ANSA is an example. This platform supports the development and construction of distributed applications independently of the existing machines in which they will run.

Based on these two tools, our goal was to make possible a quick creation of services, translating the files generated by GEODE, at the moment a service is created, in to files supported by ANSA for subsequent distribution.

In this context, a presentation about services, specifically multimedia services, and a description of tools and facilities of the two software packages previously referred is done is this thesis. An implementation for the intended translation is also proposed.

Keywords

Multimedia	*Database*	*Distributed Systems*
Formal Specification	*Object Oriented*	

Agradecimentos

Desejo aqui expressar os meus agradecimentos ao Prof. Doutor Eurico Manuel Elias de Morais Carrapatoso, meu orientador, pela sua permanente disponibilidade em proporcionar as melhores condições para a evolução deste trabalho e pelas revisões e sugestões efectuadas.

À U.T.A.D. pelas condições de trabalho proporcionadas.

Ao Ricardo e ao Benjamim pela disponibilidade e pelos esclarecimentos que muito contribuíram para este trabalho.

Aos meus colegas da U.T.A.D. pelo bom ambiente de trabalho e pela permanente disposição em ajudar.

Finalmente, um agradecimento especial à minha família pela infinita paciência e apoio sempre demonstrados.

Paula Cristina Oliveira.

Porto, Setembro de 1995

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As redes de comunicações sofreram várias e profundas transformações tecnológicas nos últimos anos, tanto no que diz respeito à transmissão como à comutação. Os recentes avanços em comunicações e o rápido desenvolvimento de sistemas computacionais, cada vez com maiores capacidades e maiores velocidades de processamento, permitem aos operadores de rede oferecer uma grande variedade de serviços estáveis, seguros e de elevado desempenho.

Tecnologias de transmissão e de comutação, como por exemplo o modo de transferência de informação ATM (*Asynchronous Transfer Mode*) [Nunes92], começam a estar disponíveis de forma a suportar serviços de dados, de voz, transmissão de imagens fixas e com movimento e, com particular destaque, aplicações multimédia que podem combinar componentes de serviço de dados, de voz e de imagem.

Definição do Problema

Os sistemas de comunicações são complexos e encontram-se numa fase de transição tornando-se cada vez mais computorizados. Avanços tecnológicos, como por

exemplo o aparecimento de processadores (CPU - *Central Processing Unit*) mais potentes e o aparecimento de redes locais (LAN - *Local Area Network*) permitindo a ligação de centenas de máquinas, tornaram possível interligar, através de redes de alta velocidade, sistemas computacionais constituídos por um grande número de CPUs. Estes sistemas são habitualmente designados por sistemas distribuídos em contraste com os sistemas que existiam anteriormente constituídos por um só CPU, a memória e alguns periféricos.

De facto podemos considerar que a rede de comunicações, baseada em computadores, se tem vindo a tornar na maior rede de computadores distribuída. As redes de comunicações que suportam aplicações de informação distribuídas são elas próprias sistemas de informação distribuídos, geridos por entidades computorizadas.

As redes modernas de banda larga são associações de servidores e utilizadores de informação, protocolos de transferência e facilidades e dispositivos de processamento. Torna-se cada vez mais óbvio que estas são sistemas distribuídos, à medida que os seus componentes se tornam mais "inteligentes" e são introduzidas unidades de computação mais pequenas em sistemas amplos.

Consequentemente, não é surpresa que técnicas de especificação formal estejam cada vez mais a ser aplicadas para ajudar a enfrentar esta complexidade. O uso de linguagens de especificação formal torna possível analisar e simular soluções alternativas, o que na prática é impossível quando se usa uma linguagem de programação devido aos custos e atrasos. Uma linguagem de especificação oferece ao utilizador um conjunto de conceitos bem definidos que melhoram a sua capacidade de produzir e justificar uma solução para um dado problema.

Com o aparecimento da rede digital de banda larga, novos serviços vão surgindo capazes de satisfazer um leque variado de utilizadores. A utilização de rede por esses serviços depende de dois factores chave: a diversidade dos serviços fornecidos e o seu baixo custo. Quanto maior for a diversidade de serviços existentes e menor for o custo de utilização desses serviços, maior será a probabilidade de um utilizador encontrar um serviço que justifique a sua ligação.

Por outro lado, com o aumento da complexidade dos sistemas e da competitividade dos mercados, torna-se cada vez mais necessário criar serviços de uma forma rápida, com menor custo e que apresentem um nível elevado de qualidade.

Objectivos

Podemos afirmar que estamos numa nova era das comunicações, sentindo a necessidade cada vez maior de uma criação rápida de serviços. É nesta área que se insere este trabalho.

Devido ao facto das comunicações caminharem no sentido de proporcionar a qualquer cidadão acesso a informação multimédia, o estudo de serviços multimédia foi um dos objectivos deste trabalho.

Sendo o SDL (*Specification Description Language*) uma linguagem que permite descrever e especificar serviços de uma forma rápida e o ANSA (*Advanced Network System Architecture*) uma arquitectura para ODP (*Open Distributed Processing*) que suporta o desenvolvimento e construção de aplicações distribuídas, o estudo dos pacotes de software GEODE (baseado no formalismo SDL) e ANSAware (implementação exemplo da arquitectura ANSA) tornaram-se, naturalmente, objectivos deste trabalho.

Finalmente, e baseado nos estudos anteriormente referidos, pretendeu-se desenvolver uma forma de criação rápida de serviços de comunicação em ambientes distribuídos, traduzindo os ficheiros gerados pelo GEODE, aquando da especificação de um serviço, em ficheiros suportados pelo ANSAware, para a posterior distribuição.

Estrutura da Dissertação

Esta dissertação encontra-se estruturada em cinco capítulos para além deste capítulo introdutório que apresenta os objectivos que guiaram o trabalho desenvolvido, bem como a estrutura do relatório.

O segundo capítulo visa situar o leitor na área dos serviços de comunicação, mais concretamente dos serviços multimédia. Desta forma, neste capítulo é feita uma descrição das características das comunicações multimédia, seguida de uma apresentação de uma classificação dos serviços de comunicação dirigidos ao tratamento de informação multimédia e finalmente são descritas algumas arquitecturas para serviços de acesso deste tipo.

No terceiro capítulo é feita uma descrição do software GEODE, um ambiente SDL desenvolvido pela VERILOG, e utilizado neste trabalho para descrever e especificar serviços de comunicação. O processo adoptado para a sua exposição consiste numa descrição geral das ferramentas existentes, seguida de uma apresentação

resumida dos conceitos mais importantes da notação SDL e, finalmente, a descrição da forma como é gerado o código de uma aplicação.

No quarto capítulo é apresentado o software ANSAware, plataforma usada para a distribuição posterior dos serviços. Neste capítulo são descritas, de uma forma resumida, as ferramentas e facilidades disponíveis nesta plataforma distribuída.

No quinto capítulo é apresentado um protótipo para a tradução pretendida. Este capítulo começa por apresentar as várias alternativas possíveis para a correspondência desejada entre elementos SDL e componentes ANSA, seguidamente é feita uma exposição da correspondência usada. O capítulo termina com a descrição da forma de tradução dos ficheiros gerados pelo GEODE em ficheiros suportados pela plataforma ANSAware.

No sexto capítulo, e com base na exposição apresentada no capítulo anterior, é apresentado um exemplo de criação de um serviço simples de *News Multimedia*.

Finalmente no sétimo capítulo é feita uma análise do trabalho efectuado e são ainda apresentadas perspectivas de evolução futura do protótipo apresentado.

Capítulo 2

Serviços de Telecomunicações

As redes públicas de telecomunicações, cuja existência é de pouco mais de um século, sofreram muitas e profundas transformações tecnológicas nos últimos anos.

Simultaneamente com o aumento do número de utilizadores e com o aumento da área geográfica abrangida, as redes de telecomunicações foram-se diversificando e especializando, à medida que novos serviços foram surgindo [Nunes92].

Actualmente, as redes públicas de telecomunicações oferecem um conjunto vasto de serviços de banda estreita, com comutação de circuitos ou comutação de pacotes como por exemplo de voz, dados, fax, correio electrónico, etc.. O conjunto de serviços disponíveis com o aparecimento da banda larga é ainda mais diverso, incluindo os serviços anteriores mas com versões de maior velocidade bem como novos serviços como por exemplo HDTV (televisão de alta definição), vídeo interactivo em tempo real, e outros que só são possíveis com as características da banda larga.

A futura RDIS-BL (Rede Digital com Integração de Serviços de Banda Larga) será uma rede de telecomunicações que integrará todos os serviços [Fig. 2.1], utilizando um novo modo de transferência de informação, conhecido como Modo de Tranferência Assíncrono (*Asynchronous Transfer Mode*) e designado por ATM.

Com a rede de telecomunicações de banda larga e relativamente às redes de banda estreita, vamos ter um aumento no volume de informação transmitida, um aumento da velocidade e a possibilidade de visualização de informação transmitida em

tempo real, como por exemplo nos serviços com imagens e vídeo. Os termos imagem e vídeo serão utilizados como abreviaturas de imagem fixa e de imagens com movimento, respectivamente.

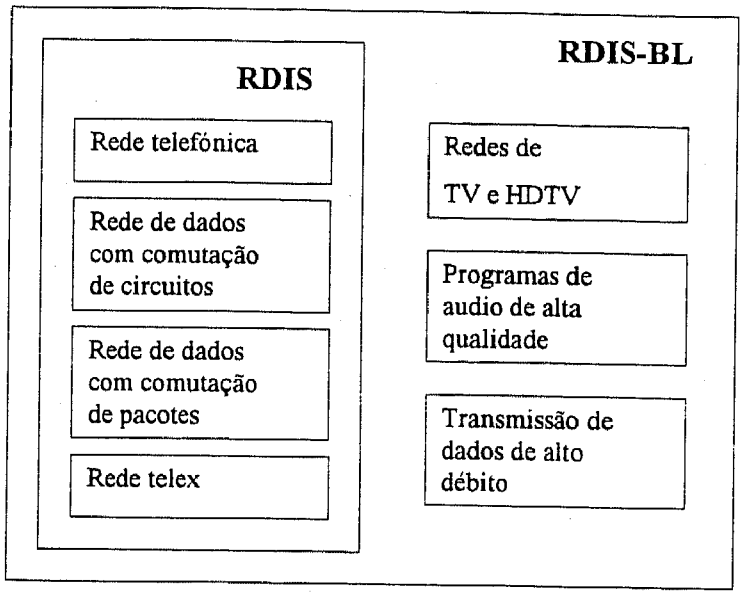


Fig. 2.1 - Integração de redes dedicadas na RDIS e RDIS-BL

Podemos descrever os serviços de banda larga de acordo com os média de comunicação: fax, dados, audio, imagem, e vídeo. Os serviços que possuem uma combinação desses média são referidos como serviços multimédia [Wright93].

Multimédia é uma das formas mais inovadoras do uso de uma rede de comunicações para obter comunicações entre pessoas e para aceder a informação.

Muitas pessoas sentem que a informação será dominada por sistemas multimédia que, como se espera, suportem todos os tipos de média, combinando-os e integrando-os como seja necessário para aplicações particulares.

Espera-se que, de uma forma crescente, os sistemas multimédia se venham a tornar importantes nos próximos anos e que corram em arquitecturas completamente distribuídas onde dados multimédia são armazenados em diferentes locais, e aos quais os utilizadores podem aceder através de uma rede.

2.1 Comunicações Multimédia

Comunicações multimédia é um campo que se refere à representação, armazenamento, acesso e divulgação de informação expressa em múltiplos média, como texto, voz, imagem, audio e vídeo [Mouftah92].

O número de aplicações multimédia existentes até à data já é razoável, mas espera-se que aumente de uma forma rápida nos próximos anos. Aplicações em medicina, educação, viagens, bancos, segurança, administração e edição estão a aparecer de uma forma rápida. Estas aplicações são caracterizadas por um grande conjunto de documentos multimédia que devem ser trocados com atrasos muito pequenos.

As comunicações multimédia podem ser definidas como uma combinação de quatro ingredientes chave [Wright93]:

- *Dois ou mais dos cinco média* de comunicação (audio, dados, fax, imagem e vídeo);

- *Sincronização* - os sistemas multimédia requerem dois tipos de sincronização entre os diferentes média: contínua e espacial. A primeira é necessária quando os média estão envolvidos simultaneamente na apresentação de informação [Heijenk94]; como exemplo temos o caso do audio/vídeo onde é necessário este tipo de sincronização para coordenar os movimentos dos lábios com o som. A segunda é requerida pelo projectista do sistema para organizar a apresentação do material da forma desejada;

- *Comunicações com utilizadores* - os sistemas multimédia envolvem comunicações com utilizadores humanos como emissores ou receptores de informação, ou ambos;

- *Capacidades interactivas* entre as partes comunicantes.

Um exemplo típico de multimédia é a videoconferência, combinando dois tipos de média (audio e vídeo), utilizadores humanos, sincronização e interacção.

Existem cinco tipos de interacções possíveis nos sistemas multimédia, que se passam a descrever:

1 *Search and browse*: corresponde a funções de acesso a bases de dados para procurar a informação multimédia desejada.
2 *Interactive buttons*: exemplos destes botões são as listas de opções, menus e partes da informação que podem ser seleccionadas para a obtenção de informação adicional.
3 *Windowing*: uma trama de informação sobrepõe-se a uma trama de fundo; o tamanho das tramas podem ser alterados pelo utilizador de forma a que se possa visualizar simultaneamente partes de algumas tramas.
4 *Fast Forward, rewind, pause e skip*: os comandos de um leitor de cassetes para controlar a sequência e a velocidade de apresentação da informação podem ser usados não só em informação de audio e vídeo mas também para controlar sequências de tramas de dados, fax e imagem.
5 *Conversational interaction*: quando nas extremidades de um canal de comunicação estão presentes utilizadores humanos, um conjunto de interacções de conversação são possíveis, como por exemplo discussões verbais, visualização de dados e imagens partilhados nos ecrans, edição, etc.

2.2 Classificação dos Serviços

Os serviços de banda larga são de muitos tipos e com diferentes requisitos de desempenho na perspectiva do utilizador. Existem várias formas de classificar este conjunto vasto de serviços de comunicação. Uma das formas de classificação é de acordo com o média utilizado: audio, dados, imagem, vídeo ou fax. Outra forma é fornecida pelo CCITT (*International Consultative Committee on Telephone and Telegraph*) [Wright93].

Este comité começa por distinguir dois grupos de serviços:

* *serviços interactivos* - envolvendo transmissão nos dois sentidos;

* *serviços de distribuição* - envolvendo transmissão num só sentido, sem transmissão de retorno.

Estes dois grupos de serviços são ainda subdivididos em vários tipos de acordo com os seus atributos.

Uma vez que um dos ingredientes chave das comunicações multimédia é a interactividade entre as partes comunicantes, os serviços de distribuição não serão aqui analisados pois não se inserem no objectivo deste capítulo. Exemplos deste tipo de serviços são a difusão de TV e rádio.

Os serviços interactivos, nesta fase de classificação, são subdivididos em:

* *serviços de mensagem* - a informação é enviada a uma base de dados intermédia e subsequentemente extraída pelo receptor. Como exemplo deste tipo de serviço temos o serviço de mensagens original, *electronic mail*, que rapidamente foi seguido por redes de fax *store-and-forward* e sistemas de mensagem de voz. Tratando-se de comunicações multimédia estes serviços envolvem composição de mensagens multimédia, armazenamento intermédio, e subsequente recuperação.

* *serviços de acesso* - neste tipo de serviço é facultado um sistema de armazenamento computorizado e a informação é extraída tal como é pretendida pelos utilizadores. Um serviço de acesso multimédia genérico é o acesso a uma base de dados multimédia por parte de um utilizador. Este pode interagir com o sistema, não só no momento em que faz o pedido mas também durante a fase de transmissão da informação. Por exemplo, durante a recepção o utilizador pode accionar as funções *fast forward*, *rewind* e *pause*.

* *serviços de conversação* - os intervenientes na troca de informação estão simultaneamente presentes. Como exemplo de um serviço de conversação multimédia podemos citar a videoconferência.

Nas Tabelas 2.1, 2.2 e 2.3 são apresentados, segundo a classificação anterior, alguns serviços de banda larga e as respectivas aplicações. Documentos mistos significa que o documento pode conter texto, gráficos, imagem, vídeo e audio.

Tipo de informação	**Exemplos de serviços**	**Aplicações**
Vídeo e audio	Mail de vídeo	Serviço de mail para a transferência de filmes
Documentos	Mail de documentos	Serviço de mail para documentos mistos

Tabela 2.1 - Serviços de Mensagem

Tipo de informação	**Exemplos de serviços**	**Aplicações**
Texto, dados, gráficos, audio imagens e vídeo	Videotexto	Educação à distância Tele-vendas Tele-segurança
	Vídeo	Diversão Educação à distância
	Imagens de elevada resolução	Educação à distância Comunic. de imagens médicas
	Documentos	Acesso a documentos mistos
	Dados	Telesoftware

Tabela 2.2 - Serviços de Acesso

Tipo de informação	**Exemplos de serviços**	**Aplicações**
Vídeo e audio	Videotelefone	Tele-vendas Tele- educação
	Videoconferência	Tele-educação Conferências
	Transmissão de informação vídeo/audio	Transmissão do sinal de TV Diálogo vídeo/audio
Dados	Transmissão de informação digital	Transferência de dados a alta velocidade
	Tele-acção	Controlo em tempo real
Documentos	Comunicação de documentos	Transferência de documentos mistos

Tabela 2.3 - Serviços de Conversação

Comunicações multimédia de conversação são interacções em tempo real entre as partes comunicantes. Serviços de acesso e mensagem implicam uma preparação de documentos multimédia e a sua subsequente visualização por parte de um utilizador. Assim podemos dividir as comunicações multimédia em duas classes:

- Comunicações multimédia em tempo real;
- Comunicações multimédia sem requisitos de tempo real.

Muitas comunicações multimédia em tempo real baseiam-se numa discussão telefónica, sendo os outros média usados para a apresentação dos objectos em discussão. Por esta razão, o audio está sempre presente como um dos média nas aplicações em tempo real. Dependendo da combinação dos outros média existentes, surgem vários sistemas de comunicação deste tipo.

O uso de imagens em publicidade, viagens, museus, e em muitos outros tipos de actividades baseadas na visualização, levou à necessidade do uso de documentos multimédia que contêm imagens. Desta forma o meio predominante nas comunicações de documentos multimédia é a imagem.

São possíveis outras classificações para as comunicações multimédia, como, por exemplo, classificações de acordo com a estrutura do documento do ponto de vista dos modos como este pode ser procurado. A classificação entre multimédia em tempo real e multimédia sem tempo real é referida aqui devido ao seu impacto directo sobre os requisitos em telecomunicações. Para uma dada qualidade de apresentação de informação ao receptor, comunicações em tempo real requerem mais largura de banda do que o acesso a um documento. Isto porque os documentos armazenados podem ser codificados usando processamento intensivo o que leva à obtenção de códigos mais eficientes, enquanto que a codificação em tempo real deve ser executada mais rapidamente, logo com menos processamento, o que leva à produção de códigos menos eficientes.

As comunicações multimédia em tempo real são extremamente importantes para facilitar discussões entre utilizadores situados em múltiplos locais. Os documentos multimédia são os melhores meios de organizar informação em ambientes de mensagem e de acesso, podendo também ser usados em ambientes de conversação quando um utilizador envia um documento deste tipo a um ou mais dos intervenientes na conversação. As comunicações multimédia, qualquer que seja o tipo, são cada vez mais usadas tanto a nível de negócios como da educação.

Os requisitos de processamento, transmissão e apresentação que a informação multimédia apresenta não são suportados pelas arquitecturas dos sistemas de comunicação tradicionais. Uma arquitectura define as funcionalidades dum sistema de comunicação de modo a este satisfazer os requisitos dos serviços que irá suportar. Assim, a integração de aplicações multimédia num sistema requer a definição de novas funcionalidades de forma a satisfazer os requisitos das aplicações.

Na secção seguinte são apresentadas algumas arquitecturas para serviços de acesso.

2.3 Arquitecturas para Serviços de Acesso

O conceito de arquitectura pode ser aplicado a vários níveis e segundo o modelo de comunicações cliente/servidor existem três níveis a destacar no que diz respeito à informação: armazenamento, localização e transmissão [Oliveira95].

2.3.1 Armazenamento de Informação

Um Sistema Gestor de Base de Dados (SGBD) tem como funções armazenar, gerir, aceder e explorar, no ponto de vista de apresentação, toda a informação guardada na Base de Dados.

No que diz respeito ao armazenamento, o SGBD deve suportar mecanismos específicos quando se trata de informação multimédia, pois este tipo de informação possui características próprias, como, por exemplo, dependência temporal, estrutura complexa e grande volume de dados.

Existem actualmente quatro mecanismos, que podem ser utilizados pelos SGBDs [Rakow95], para gerir informação multimédia: referências externas, campos longos, uso de funções externas e orientado a objectos.

Referências Externas

O uso deste mecanismo implica que toda a informação multimédia original seja referenciada. As referências podem ser, por exemplo, os nomes dos ficheiros ou qualquer outro identificador que localize os dados. Além das referências, as bases de dados contêm informação adicional sobre relações e atributos, que correspondem a

informação adicional sobre a informação multimédia armazenada, podendo ser, por exemplo, o tamanho, o formato de compressão, o dispositivo de saída ou uma descrição do conteúdo da informação.

Campos Longos

Um campo longo (*BLOB - Binary Large Object*) pode armazenar informação até alguns Gbytes, sendo no entanto possível aceder a parte da informação aí armazenada [Shetler90].

Uso de Funções Externas

Este mecanismo permite a chamada de funções externas para processamento de dados. Os SGBDs que usam este tipo de mecanismo podem executar chamadas a funções externas, mas não podem controlar a sua execução. Por exemplo, se é usada uma função externa para o envio de uma sequência de audio, não pode ser garantido pelo SGBD o acesso exclusivo. Estas funções externas são muitas vezes úteis para mecanismos de autorização, suporte multi-utilizador e reutilização de ferramentas e algoritmos existentes no que diz respeito à apresentação e aquisição de informação multimédia.

Orientado a Objectos

Este tipo de mecanismo, para além de permitir que os programadores definam tipos de dados abstractos e os referenciem nas suas aplicações, permite ainda a construção de hierarquias de tipos de dados. Desta forma os sistemas orientados a objectos oferecem o suporte mais apropriado para informação multimédia; no entanto apresentam ainda algumas deficiências no que diz respeito à dependência temporal, controlo e indexação, isto é, suporte para informação dependente no tempo, interacção com o utilizador e acesso/perguntas baseadas nos conteúdos.

Os SGDBs devem suportar um modelo de dados orientado aos objectos, sendo este modelo constituído pelos seguintes elementos [Rumbaugh91]:

- objectos com identidade;
- atributos e métodos;
- classes;
- hierarquia e herança de classes.

Neste modelo as entidades são representadas por objectos.

Cada objecto possui mais dois atributos para além da identidade: um valor e um comportamento. Associado ainda a cada objecto estão os métodos, que definem o seu comportamento.

Os construtores de tipos permitem construir, no caso de informação multimédia, objectos com uma estrutura bem definida compostos por vários média. Um objecto pode referenciar outros objectos através das suas identidades.

Uma classe é um conjunto de objectos com as mesmas características estruturais ou comportamentais [Kretz92]. A existência de especializações de uma classe, originando assim sub-classes leva, à noção de hierarquia de classes, herdando estas sub-classes as propriedades da classe origem. De igual modo, podem existir generalizações de um grupo de classes dando origem a uma super-classe.

Na Fig. 2.2 é apresentada, segundo este modelo, uma possível estrutura de uma base de dados de pousadas de Portugal.

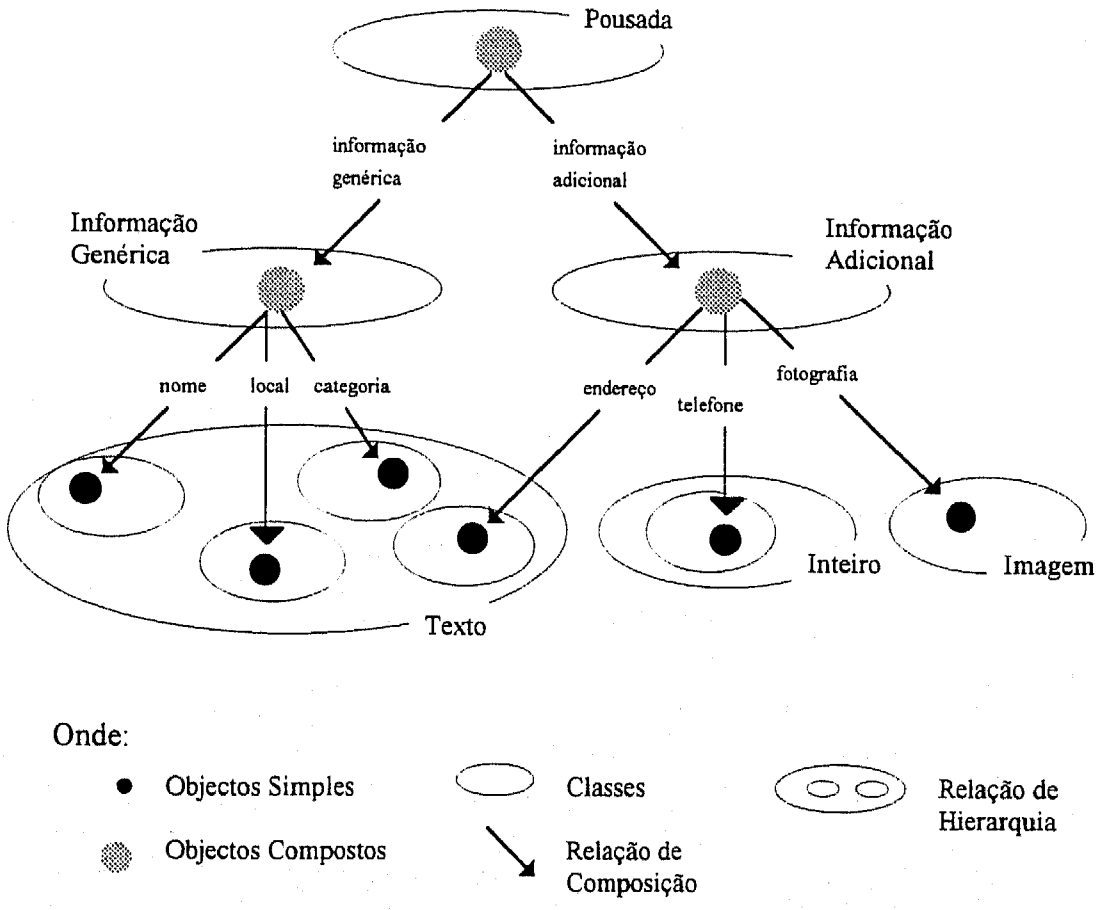


Fig. 2.2 - Estrutura orientada a objectos de uma Base de Dados de Pousadas de Portugal

2.3.2 Localização da Informação

Num serviço de acesso a informação, é de extrema importância a localização dessa informação. Esta deve estar armazenada de uma forma segura, organizada e funcional, permitindo que as várias aplicações possam aceder à informação de uma forma eficiente. O sistema de armazenamento torna-se desta forma num dos principais blocos de um serviço de acesso. O sistema de armazenamento pode ser constituído por vários subsistemas, interligados por uma rede de comunicação. Estes últimos podem ser classificados segundo a sua localização, como podemos observar na Fig. 2.3.

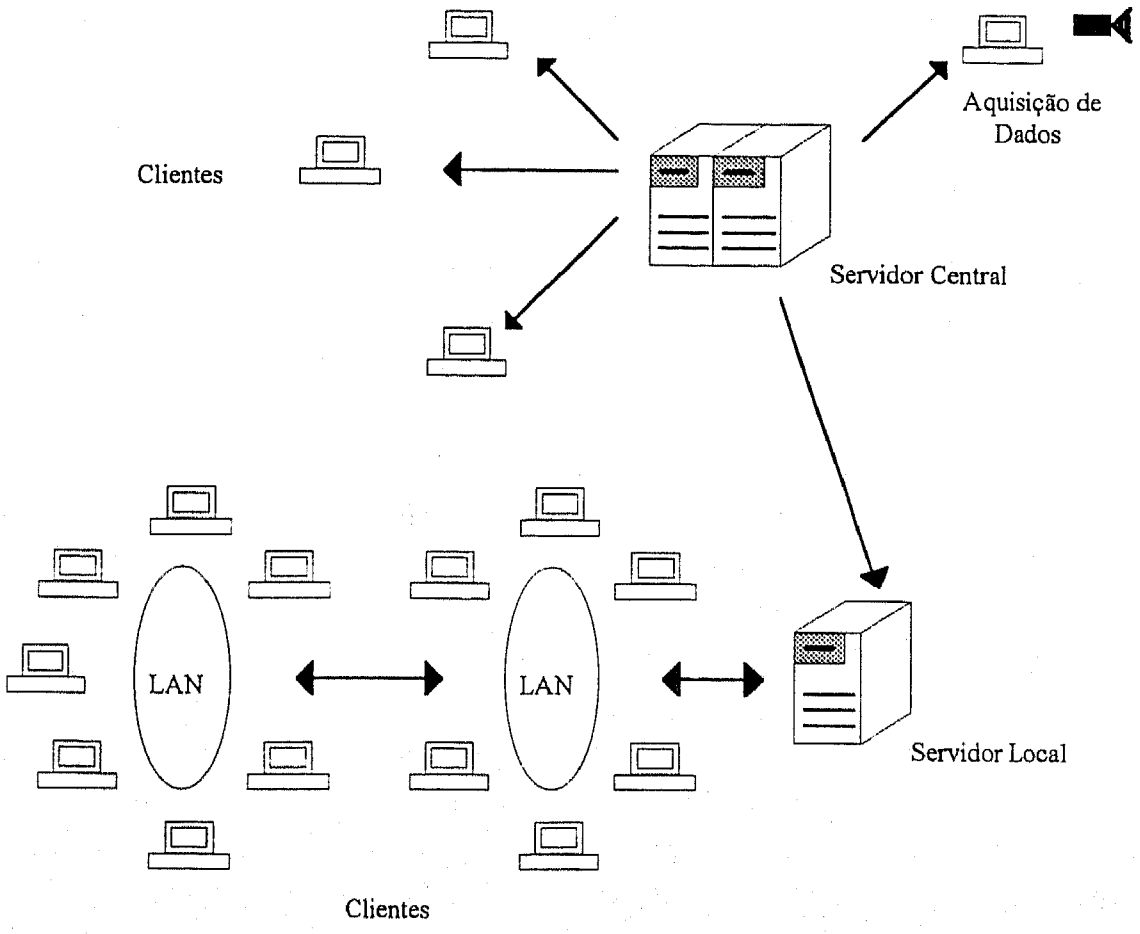


Fig. 2.3 - Sistema de armazenamento em rede

Neste tipo de sistema toda a informação está armazenada no servidor central. O uso ou não de servidores locais depende do número de utilizadores (clientes) que pretendam aceder ao servidor central. Se o número de clientes for elevado, implicando um grande volume de informação transmitida, a ligação directa ao servidor central pode tornar-se ineficiente no que diz respeito ao tempo de resposta, principalmente se se tratar de informação multimédia. Este tipo de problema pode ser solucionado através do uso de servidores locais, responsáveis pela distribuição de informação a um conjunto de clientes. Para tal este servidor está localizado perto dos clientes e possui uma cópia da informação do servidor central mais requerida por parte destes clientes. A actualização desta informação pode ser realizada periodicamente, quer através da rede de comunicações, quer através de dispositivos fisicos de armazenamento.

O uso destes servidores locais faz com que o tempo de resposta seja menor, devido ao encurtamento da distância entre os clientes e o servidor, e com que a produtividade de um grupo de clientes seja significativamente incrementada, não somente devido à não necessidade de ligações de rede mas também à resposta do servidor central, e finalmente faz com que haja uma redução no custo global das comunicações.

Existe ainda outro tipo de subsistema de armazenamento, baseado em dispositivos de armazenamento localizados nas estações de trabalho dos clientes. Estes são utilizados não só como suporte básico para as aplicações que estão a correr como também são utilizados como *cache* para a informação fornecida por qualquer dos servidores referidos anteriormente.

Os equipamentos usados por estes subsistemas de armazenamento depende do tipo e da quantidade de informação a armazenar. Como exemplo, consideremos um servidor multimédia de vídeo. O armazenamento de um filme de 100 minutos de duração, à taxa de 0,5 Mbytes/s para vídeo em tempo real, requer 3 Gbytes de espaço em disco. Para armazenar 1000 destes filmes, é requerida uma capacidade de disco de 3 Tbytes [Rangan92]. Desta forma, um servidor deste tipo de informação requer dispositivos de elevada capacidade de armazenamento.

Nos servidores central e locais são normalmente utilizados jukeboxes de discos ópticos enquanto que nos clientes se utilizam dispositivos de armazenamento com capacidades que vão desde 1,4 Mbytes (disquetes) até 5 Gbytes (tapes, utilizadas normalmente para arquivos de informação e para backups).

Na Fig. 2.4 são apresentados alguns tipos de informação e os seus respectivos requisitos no que diz respeito ao armazenamento [Cole93].

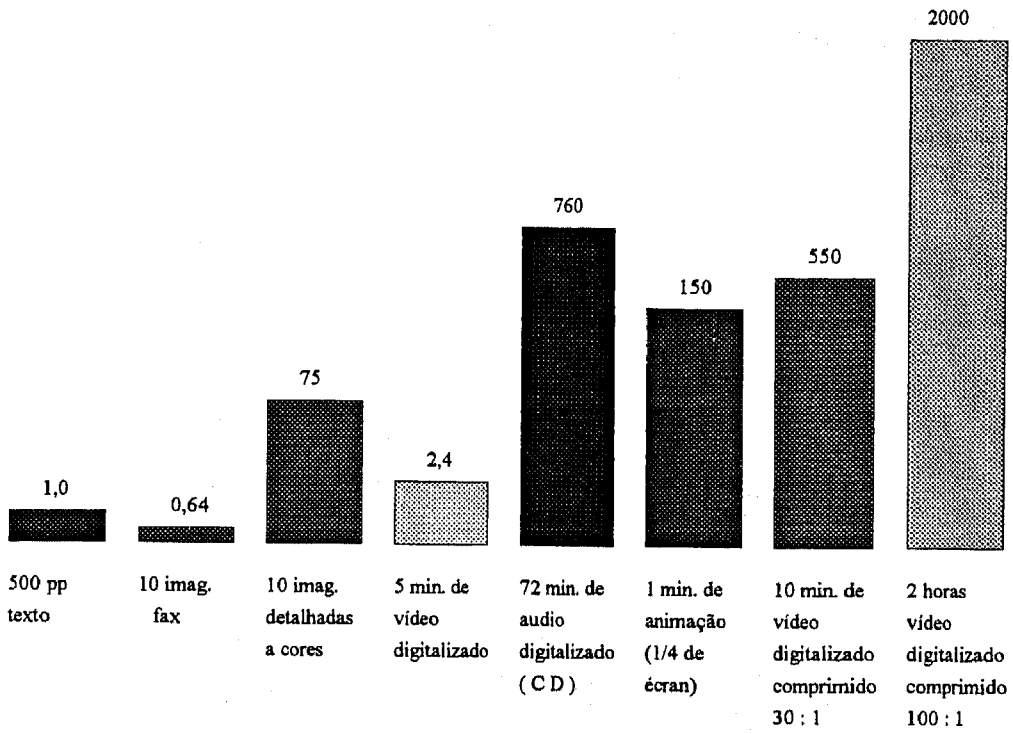


Fig. 2.4 - Requisitos de Armazenamento Multimédia

2.3.3 Transmissão de Informação

Num serviço de acesso remoto existem três intervenientes essenciais: o servidor, a rede e o cliente [Fig. 2.5]. Devido à diversidade de equipamentos existentes torna-se necessária a existência de normalização no que diz respeito à representação da informação nos sistemas onde esta é armazenada, transmitida e visualizada.

Como exemplos temos o JPEG (*Joint Photographic Experts Group*), formato para armazenamento, transmissão e visualização de imagens paradas com tonalidades contínuas, o MPEG (*Motion Pictures Experts Group*), usado para a compressão de audio e vídeo digitais sincronizados, e o MHEG (*Multimedia and Hypermedia Experts Group*), para transmissão de informação multimédia e hipermédia. Pode-se definir o conceito informação hipérmedia como sendo informação multimédia que contém ligações a outra informação multimédia.

Outro factor importante a ter em conta são as capacidades dos intervenientes no serviço; por exemplo, se a rede sobre a qual está implementado um serviço não tiver

capacidade suficiente para transferência de informação em tempo real, este tipo de transferência não poderá ser efectuada.

São várias as arquitecturas possíveis que um serviço de acesso pode ter. Cada arquitectura define, segundo o modelo de comunicações [Fig. 2.5], a localização do armazenamento e do processamento da informação, o que deverá ser efectivamente transmitido e de que forma, se em tempo real ou não.

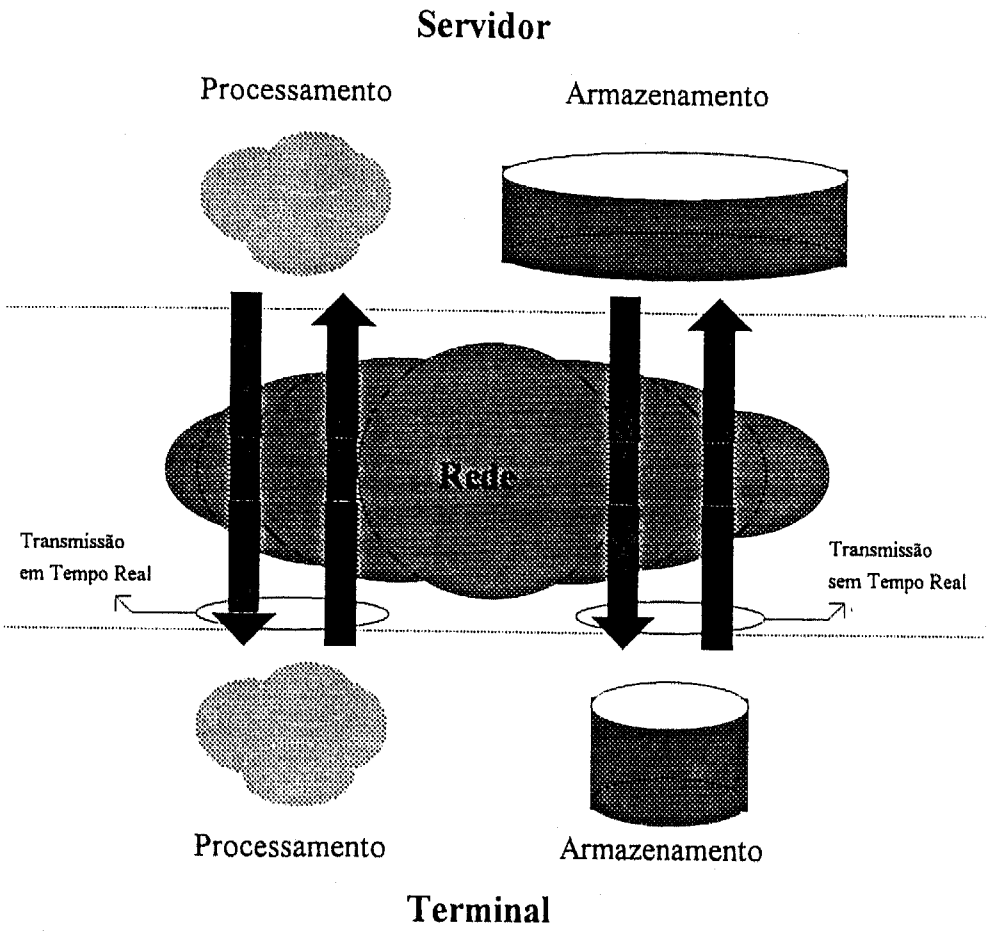


Fig. 2.5 - Modelo de Comunicação

Nas redes de elevada velocidade, tem sido recomendado como veículo de transporte o Modo de Transferência Assíncrono (ATM).

A técnica ATM é suficientemente flexível e versátil para o fornecimento de serviços capazes de integrar um grande conjunto de fontes de tráfego multimédia, tais como dados, voz e vídeo [Habib92]. Estas fontes apresentam diversas características de tráfego com diferentes correlações e parâmetros.

Nestes casos, torna-se pertinente desenvolver um conjunto de parâmetros simples que melhor caracterizem a variabilidade e as correlações estatísticas do processo de chegada de pacotes. Estes parâmetros são então usados pela rede para alocar os seus recursos aos novos utilizadores de modo a evitar a congestão e a manter a qualidade de serviço (QoS - *Quality of Service*) pré-definida para cada utilizador. A QoS é um conjunto de parâmetros de medida de desempenho, tais como a máxima taxa de perda de informação aceitável e a variabilidade do tempo de atraso, requeridos para manter uma qualidade aceitável para o utilizador [Habib92]. Como os diferentes tipos de tráfego têm diferentes medidas de desempenho, cada tipo de tráfego tem a sua própria QoS.

Como referido anteriormente, o tráfego multimédia pode ser de duas categorias: em tempo real ou não. O tráfego em tempo real requer tempos de atraso que não excedam um dado valor, mas pode tolerar uma pequena taxa de perdas de células. O tráfego sem tempo real pode tolerar tempos de atraso variáveis mas não pode tolerar perda de células.

Capítulo 3

GEODE: Um Ambiente SDL

Com o aumento da complexidade dos Sistemas e da competitividade dos mercados, os projectistas de software e de Sistemas em Tempo Real deparam-se com um difícil desafio: como desenvolver sistemas mais rapidamente, com menor custo, que ofereçam o nível de qualidade desejado e possam ser sujeitos a posteriores alterações.

Com o GEODE, um ambiente SDL (*Specification and Description Language*) desenvolvido pela VERILOG, que suporta o ciclo de desenvolvimento desde o projecto à geração de código de aplicações, pode-se em certa medida dar resposta a este desafio multi-dimensional. O GEODE integra um conjunto de ferramentas que suportam edição sintáctica e gráfica, verificação semântica, simulação e geração de código.

Os campos de aplicação preferenciais do GEODE são, entre outros, os Sistemas Distribuídos e os Sistemas em Tempo Real [Encontre89].

Um Sistema é dito em Tempo Real quando reage a solicitações aleatórias do seu ambiente (controlo e informação) num dado tempo limitado. E é dito Distribuído quando é dividido em subsistemas autónomos, executados concorrentemente, e que cooperam para a realização de um objectivo comum.

Tais Sistemas encontram-se em muitos sectores da Indústria e, mais particularmente, em Telecomunicações, Defesa, Aeronáutica, Transporte ou Controlo de Processos.

3.1 GEODE - Descrição Geral

O GEODE é membro da família CASE (*Computer Aided Software Engineering*). É baseado no formalismo SDL e contempla várias fases do ciclo de desenvolvimento de Sistemas em Tempo Real e Distribuídos, sendo composto por uma série de ferramentas, como se pode ver na Fig. 3.1.

As notações SDL (*Specification and Description Language*) e MSC (*Message Sequence Chart*), normalizadas pela ITU-T (*International Telecommunication Union* previamente CCITT), respectivamente nas recomendações Z.100 (SDL-88) e Z.120 (MSC-92), tornaram-se cada vez mais populares na área das Telecomunicações.

O desenvolvimento do SDL teve início em 1972 depois de um período de investigação. A primeira versão da linguagem foi emitida em 1976, seguida por novas versões em 1980, 1984 e 1988. As duas últimas versões contêm uma expansão considerável da linguagem, a última das quais é considerada como a versão que atingiu um estado maduro e estável [Belina91]. Hoje em dia, o SDL é conhecido no campo das Telecomunicações, mas tem uma área de aplicação mais vasta, cobrindo Sistemas em Tempo Real, Interactivos e Distribuídos. A sua popularidade no campo das Telecomunicações deveu-se em grande parte ao facto do SDL ter sido desenvolvido por pessoas com grande experiência no projecto e na especificação de Sistemas nesta área.

Uma aplicação SDL é uma estrutura hierárquica de componentes interligados por canais que transportam sinais. O nível de decomposição mais baixo é constituído por máquinas de estados que comunicam entre si e que são chamadas *processos*. A informação processada ou trocada pode ser complexa sendo, nesses casos, descrita recorrendo a ADTs (*Abstract Data Types*). Os ADTs descrevem as propriedades funcionais de objectos de dados. Um ADT define o resultado de operações em objectos de dados sem entrar em pormenor de como esse resultado é obtido. Podemos definir um ADT como sendo um conjunto de tipos de dados, que estão relacionados uns com os outros através de operações, podendo estas últimas conter argumentos de diferentes tipos.

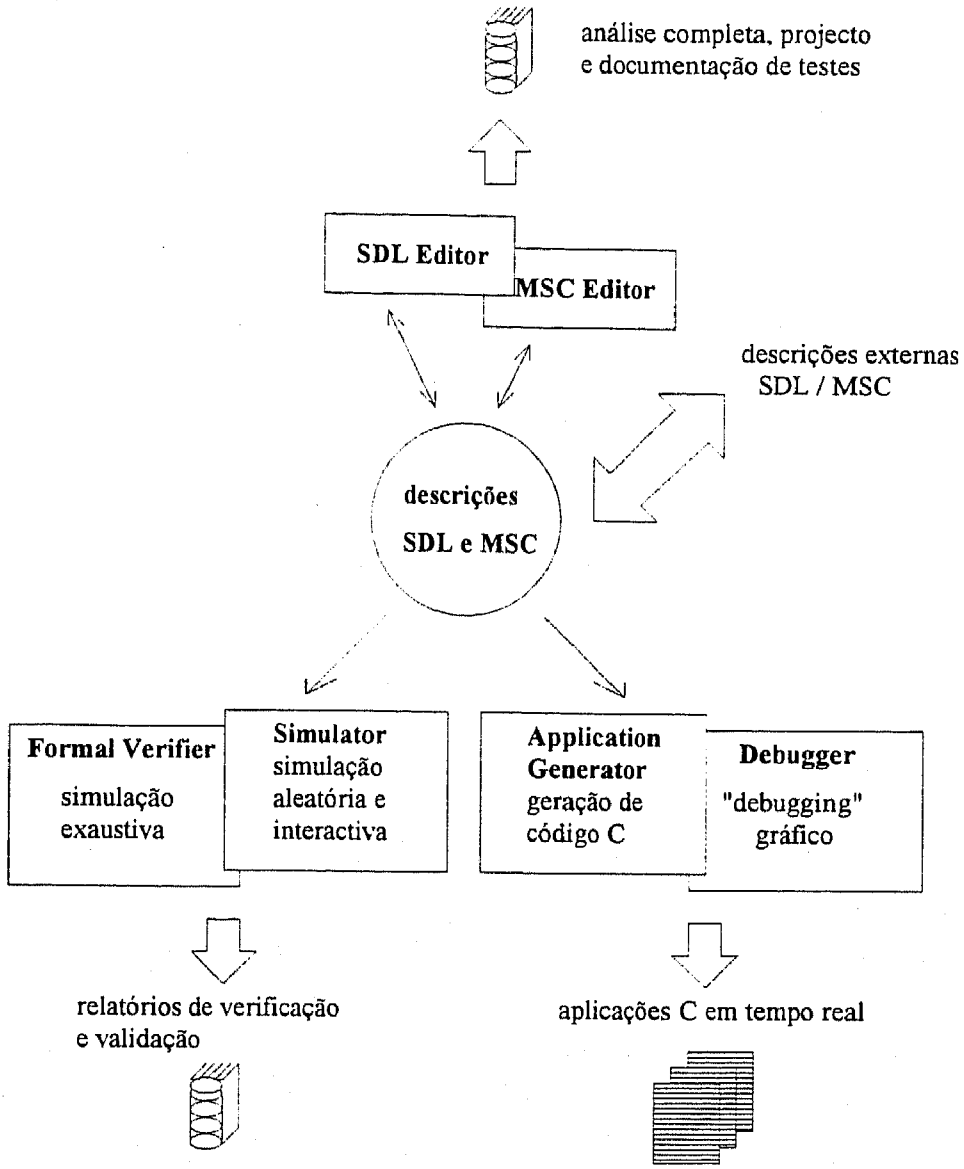


Fig. 3.1 - Ferramentas GEODE

Com a notação MSC, a ITU-T formalizou um meio de descrever a troca de informação entre componentes que hoje em dia já é usada por milhares de engenheiros em todo o mundo. Dependendo da aplicação em vista, a notação MSC pode ser usada para (i) expressar requisitos para serem satisfeitos por uma aplicação SDL, (ii) definir sequências de teste, ou (iii) delinear a execução da aplicação [VERILOG93a].

Cada uma destas duas notações apresenta duas sintaxes (equivalentes) [VERILOG93a]:

* gráfica, para ser suportada por ferramentas gráficas em *workstations* (SDL/GR, MSC/GR),
* textual (SDL/PR, MSC/PR).

Editores SDL e MSC

O GEODE contém dois editores que podem comunicar entre si, dedicados às notações SDL e MSC. Ambos são *multi-window* e *multi-view* [VERILOG93a], o que significa que os utilizadores podem abrir simultaneamente várias janelas da mesma descrição, contendo diferentes níveis de decomposição da aplicação, e qualquer modificação numa *view* é imediatamente propagada para as outras *views*.

O Editor SDL gere quatro tipos de diagramas: decomposição hierárquica de componentes, interligação, máquina de estados finitos e declaração de dados. O Editor MSC gere dois tipos de diagramas: MSC, usados para descrever sequências de trocas de sinais entre entidades SDL, e de decomposição de cenário, que permitem descrever os possíveis cenários existentes.

Os editores também oferecem ajuda semântica: de cada vez que o utilizador deseja reusar um objecto, o editor lista todos os objectos que se podem usar nesse contexto particular. Este mecanismo de partilha de dados está também disponível entre os dois editores: por exemplo, um sinal definido no Editor SDL é proposto como mensagem no Editor MSC.

Erros léxicos e sintácticos são evitados graças à interface gráfica amigável com o utilizador. As regras semânticas SDL e MSC podem ser verificadas e os erros corrigidos rapidamente.

Simulação, Verificação Formal e Validação

O GEODE inclui uma ferramenta poderosa de simulação e validação. Uma das vantagens do simulador reside nas suas capacidades para verificação e validação de especificações SDL quando comparadas com diagramas MSC ou expressões lógicas. Estas expressões lógicas não são mais do que condições de paragem para a detecção de situações catastróficas.

A ferramenta suporta vários tipos de simulação [VERILOG93d]: interactiva, aleatória e exaustiva.

No modo interactivo, o simulador oferece características avançadas tais como a execução passo a passo, que pode ser gravada e executada novamente, a escrita de comandos e o rastreio gráfico de diagramas SDL e MSC. Uma descrição SDL pode ser executada automaticamente usando o modo aleatório.

A simulação exaustiva, acedida em qualquer instante a partir da interface do simulador, oferece as seguintes capacidades:

* verificação: detecção de todos os modelos de procedimentos a serem evitados numa descrição SDL (*deadlocks*, *livelocks*, *dynamic errors*, *unspecified receptions*);
* validação: prova de que a descrição SDL fornece o serviço para que foi projectada, descrita com MSCs ou expressões lógicas.

Durante a simulação exaustiva, cenários respeitantes a erros ou violações de serviço são automaticamente armazenados e podem ser vistos posteriormente interactivamente ou convertidos em diagramas MSC.

O simulador pode ser controlado tanto a partir de uma interface gráfica, que inclui menus e a apresentação automática das descrições SDL e MSC, ou através de uma linguagem textual de comandos.

O simulador deve ser uma ajuda e não um entrave para os potenciais projectistas de sistemas complexos, devendo por isso ser possível usá-lo de uma forma simples para detectar erros no projecto. Este é o motivo pelo qual o simulador pode gerir descrições incompletas: de facto, é possível simular logo que a descrição esteja semanticamente correcta. Além disso, o utilizador não necessita de alterar a descrição SDL para atingir os objectivos do teste. Em vez disso, o simulador pode ser configurado para ignorar algumas partes da descrição.

Gerador de Aplicações

O Gerador de Aplicações pode fornecer 100% do código executável de uma aplicação multi-tarefa e distribuída a partir da descrição SDL. Também são geradas *makefiles* para tornar automática a construção de processos. O código gerado pode ser modificado e ligado a código externo.

O Gerador de Aplicações do GEODE necessita de um sistema RTE (*Real-Time Executive*) para ser executado. Contudo, o código gerado é independente do sistema destino e pode ser facilmente mapeado para outros RTEs. Isto é possível graças à

biblioteca run-time (RTL - *Run-Time Library*) fornecida no código fonte, que garante a interface entre o código gerado e o RTE destino. As bibliotecas do GEODE suportam um grande número de Executivos de Tempo Real (ou Sistemas Operativos) disponíveis no mercado, tais como Unix SV, MTOS-UX, OSE, PSOS+, VRTX32, RTC/XEC ou VMS.

3.2 Notação SDL

A notação SDL é usada para descrever um sistema como um conjunto de FSMs (*Finite State Machines*) executadas em paralelo que comunicam entre si. Uma descrição SDL é composta por: uma arquitectura hierárquica [Fig. 3.2], diagramas de interligação [Fig. 3.3], declarações de dados e procedimentos descritos por FSMs.

3.2.1 Arquitectura hierárquica

Um *sistema* em SDL é descrito por um conjunto de *blocos*, *processos* e *procedimentos* (Diagrama Hierárquico). O *sistema* interage com o exterior através de *canais*, e é constituído por *blocos* que comunicam entre si através de outros *canais*. Estes *blocos* podem ser constituídos por mais *blocos* pelas seguintes razões: por coerência com a estrutura arquitectónica da aplicação, para simplificar a descrição agrupando *blocos*, para facilitar o desenvolvimento por grupos de projectistas distintos ou o uso de projectos ou código já existentes. No nível mais baixo da decomposição hierárquica, cada *bloco* contém um ou mais *processos* que descrevem o seu funcionamento, podendo estes últimos conter *procedimentos*.

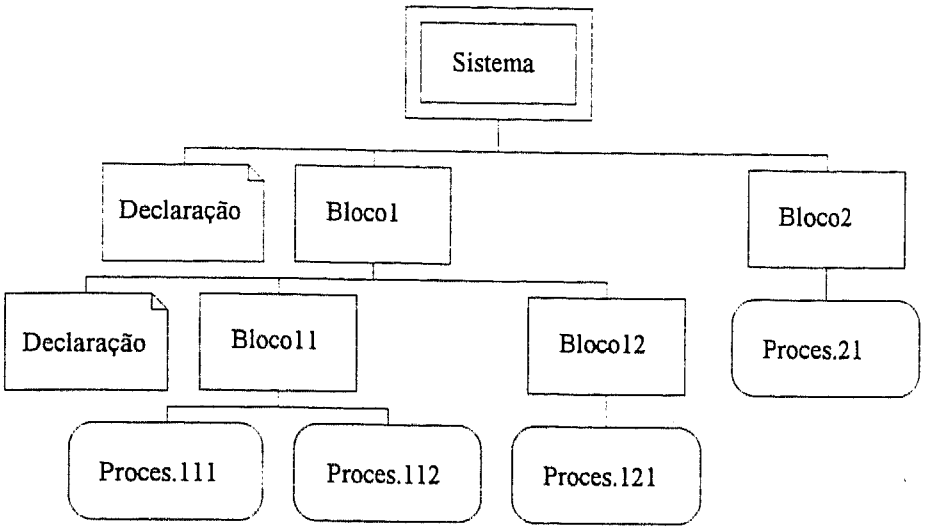


Fig. 3.2 - Diagrama hierárquico

3.2.2 Diagrama de interligação

As ligações entre as entidades estruturais que fazem parte da arquitectura do sistema são descritas usando Diagramas de Interligação (*canais* e *vias*).

Os *canais* são usados para interligar *blocos* e para a ligação com o exterior do *sistema*, e as *vias* (*signal routes*), utilizadas no nível bloco, são usadas para a interligação de *processos*. *Vias* e *canais* são conectados nas fronteiras dos *blocos* [Fig. 3.3].

Os *processos* comunicam uns com os outros transmitindo *sinais* através das *vias*. Esta transmissão de *sinais* é assíncrona, isto é, o *processo* envia o *sinal* e continua a sua execução, sem ter de esperar por qualquer resposta do receptor do *sinal*. Os *sinais* podem conter informação (*dados*). Estes *sinais* e os *tipos de dados* podem ser declarados em qualquer nível e podem ser usados, a partir da sua declaração, de acordo com a hierarquia do sistema. O uso de identificadores globais ou partilhados é estritamente controlado pelos mecanismos de exportação e de *view*, isto é, quando se cria uma especificação é construída uma lista de toda a informação existente na descrição, e quando o utilizador necessita de usar um desses identificadores, como por exemplo sinais, o editor apresenta todos os identificadores legais neste contexto e o utilizador usa simplesmente o *mouse* para seleccionar o desejado, o que força a modularidade do projecto.

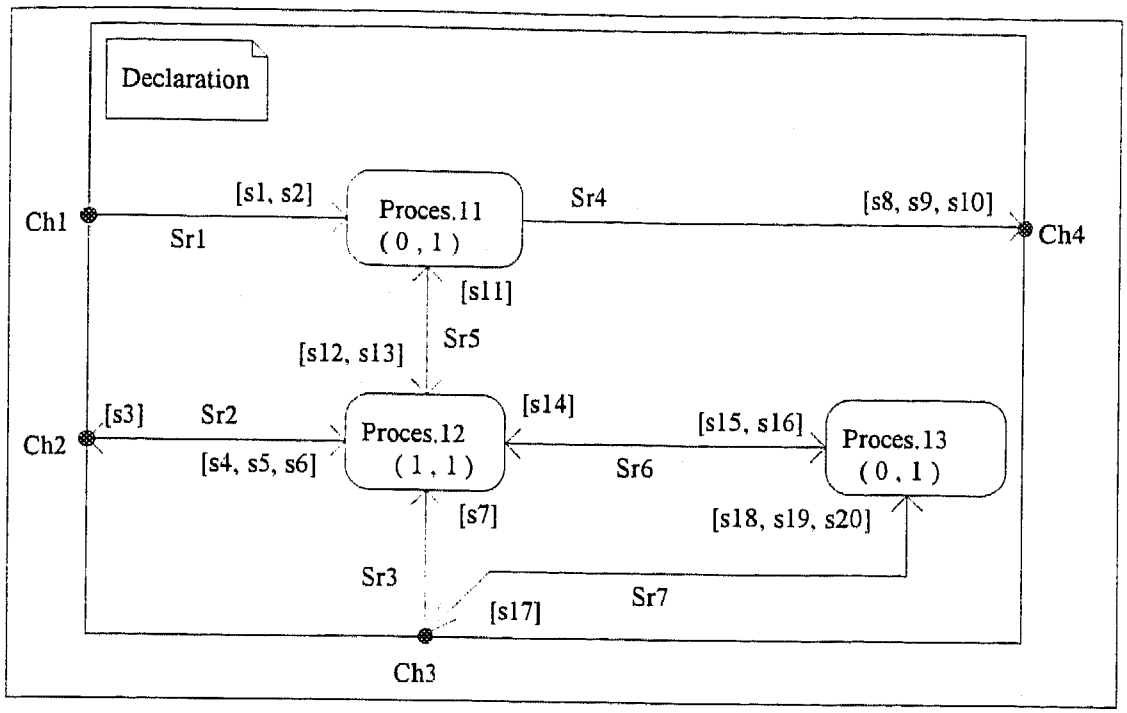


Fig. 3.3 - Diagrama de interligação

3.2.3 Declaração de dados

Em SDL, *sinais* e *dados* são definidos em símbolos especiais chamados *declarações* existentes em qualquer nível da hierarquia (mostrados nos Diagramas de Interligação bem como no Diagrama Hierárquico).

O SDL fornece um conjunto de tipos de dados pré-definidos que inclui **Integer**, **Real**, **Natural**, **Boolean**, **Character**, **Duration**, **Time**, **PId (*Process Identifier*)** e tipos complexos pré-definidos tais como **Array** e **Structure**. Uma descrição dos tipos de dados disponíveis é feita no Anexo A.

Além destes tipos de dados pré-definidos, é permitido ao utilizador definir novos tipos de dados e ADTs. A construção ADT é um conceito ***object-oriented***: é um género de "pacote" contendo dados e operadores para esses dados. O SDL suporta ainda heranças entre ADTs [Fig.3.6]. Os operadores podem ser definidos por meio de axiomas ou usados para integrar código externo às descrições SDL.

Os tipos de *dados* definidos pelo utilizador, os *sinais* e as *constantes*, podem ser declarados em qualquer nível da hierarquia, enquanto que as *variáveis*, os *parâmetros formais* e os *timers*, só podem ser declarados nos níveis *processo* ou *procedimento*.

```
NEWTYPE tipo_dado
        STRUCT
                code INTEGER;
                nome CHARSTRING;
ENDNEWTYPE tipo_dado;

DCL  pdu tipo_dado;

TIMER  rtx_time;

SIGNAL  pedido ( tipo_dado );
```

Fig. 3.4 - Declaração de dados

3.2.4 Descrição do Funcionamento

Um *processo* SDL é descrito por uma máquina de estados finitos (FSM), que define a reacção do *processo* a *sinais*, dependendo esta do seu estado interno. As acções efectuadas durante as transições são descritas por meio de recepções de *sinais*, saídas de *sinais*, *tarefas* (*tasks* - correspondendo a manipulação de *dados*), decisões, chamadas a *procedimentos*, manipulações de tempo, instanciações de *processos* e *loops*. Cada instância de *processo* tem a sua fila de entrada do tipo FIFO (*First-in, First-out*) [VERILOG93a] que permite a transmissão assíncrona referida na secção 3.3.2.

Para descrever o funcionamento dos *processos* e dos *procedimentos*, existe um conjunto de símbolos gráficos disponíveis [VERILOG93b], [VERILOG93a]: estado inicial, estado, fim, entrada e saída de sinais, decisões, tarefas, chamadas a procedimentos, criação de uma instância de processo, etc. Alguns destes símbolos podem ser visualizados na Fig. 3.5.

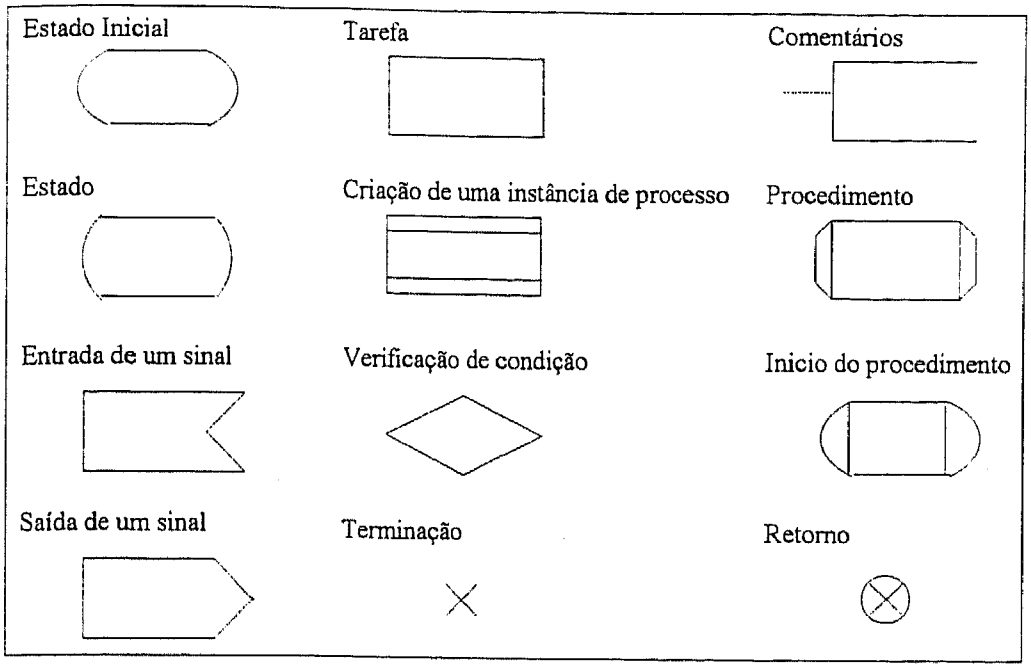


Fig. 3.5 - Símbolos gráficos SDL

Instâncias de *processos* podem ser criadas no arranque do sistema ou durante a execução, por outras instâncias de *processos*. Um *processo* pode ter várias instâncias (zero, uma ou mais), sendo cada uma delas identificada por um PId (*Process Identifier*); o PId é um tipo de dados pré-definido usado na transmissão de sinais para indicar a instância do *processo* receptor quando mais do que uma instância estão ligadas à instância transmissora.

Os operadores ADT podem ser usados nas transições de um processo [Fig. 3.6].

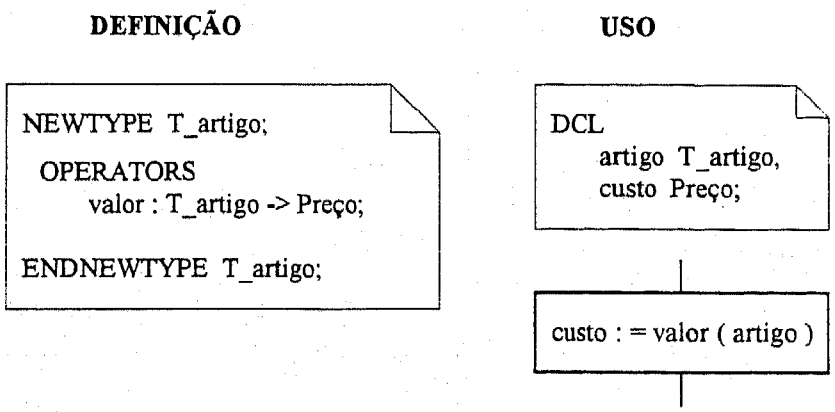


Fig. 3.6 - Definição e uso de ADTs

3.3 Gerador de Aplicações

O Gerador de Aplicações é usado para construir aplicações que funcionam em tempo real a partir de descrições SDL, que podem, opcionalmente, integrar código fornecido pelo utilizador. Estas aplicações são compostas por:

* uma biblioteca run-time dedicada ao executivo destino que é independente da aplicação e parcialmente dependente do sistema destino,
* o código gerado como resultado da tradução de SDL para C, que é independente do sistema destino,
* o código escrito pelo utilizador, que consiste essencialmente na implementação dos operadores descritos nos ADTs e em tarefas externas.

A geração de uma aplicação envolve três tipos de máquinas [VERILOG93c]:

* a **plataforma GEODE**, para o Editor, Simulador e Gerador de Aplicações, onde ficheiros fonte C, *includes* e *makefile* são gerados, a partir dos ficheiros de descrição e de configuração;

* o ***cross development host*** para compilar e linkar os ficheiros gerados e ficheiros C do utilizador para a produção de ficheiros binários das aplicações;

* a **máquina destino** onde vai ser executada a aplicação.

A plataforma GEODE e o *cross development host* podem ser a mesma máquina, partilhar o mesmo sistema de ficheiros numa rede, partilhar a mesma rede ou necessitar de um dispositivo magnético para a transferência de ficheiros [Fig. 3.7].

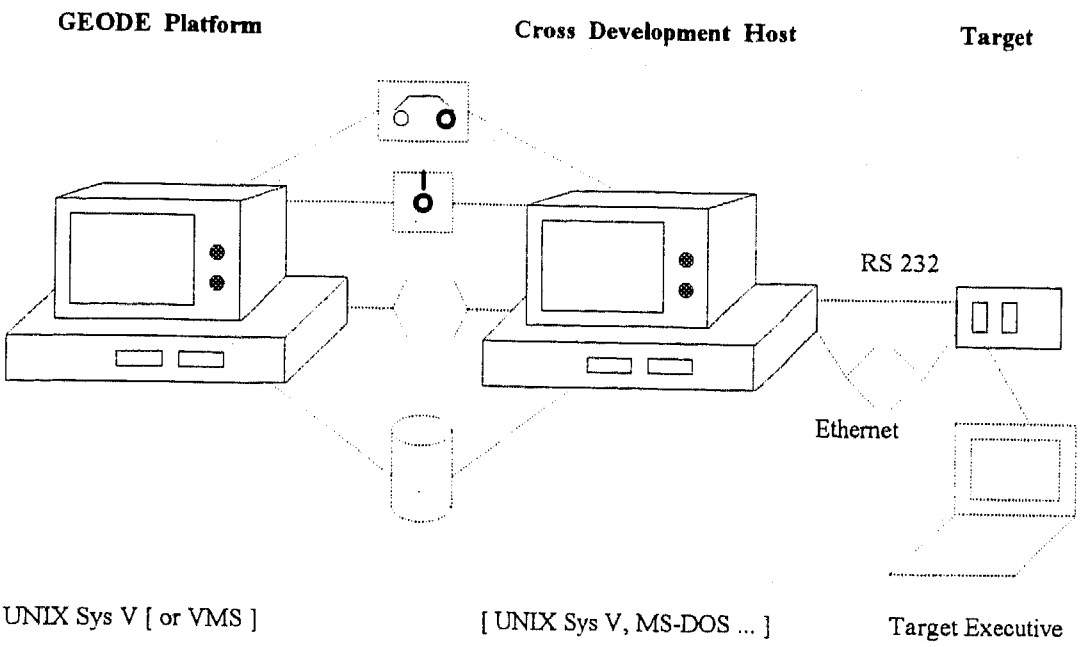


Fig. 3.7 - Máquinas envolvidas na Geração da Aplicação

A geração da aplicação é composta por quatro etapas:

1ª - Preparação da entrada para a geração;
2ª - Geração do código fonte C da aplicação;
3ª - Compilação e linkagem da aplicação;
4ª - Execução da aplicação na máquina a que foi destinada.

A terceira e a quarta etapa não serão abordadas neste capítulo, uma vez que não se inserem no âmbito deste trabalho. Estas etapas serão posteriormente realizadas na plataforma ANSA.

3.3.1 Preparação da entrada para a Geração

O Gerador de Aplicações necessita de dois ficheiros para a geração do código fonte C da aplicação:

* o ficheiro de descrição verificado (*descrição*.**gr**),
* o ficheiro de configuração (*descrição*.**cfg**).

O primeiro (*descrição*.**gr**) é construído a partir da forma textual SDL/PR usando a ferramenta **geodeverif** e o segundo (*descrição*.**cfg**) é construído pelo utilizador, em qualquer editor de texto, e descreve a arquitectura física da aplicação.

3.3.1.1 Tradução de SDL para C

Arquitectura

Uma descrição SDL é composta logicamente por uma hierarquia de objectos estruturais, sendo os *processos* os objectos que expressam a descrição do funcionamento, como referido anteriormente.

A implementação física da descrição SDL deriva desta arquitectura e consiste numa hierarquia dos seguintes objectos:

* **nó** : todo o software controlado por um sistema operativo multi-tarefa em tempo real (corresponde a um *kernel* multi-processo);
* **tarefa** : unidade de paralelismo do S. O.;
* **instância de processo** : unidade de paralelismo do SDL.

Geralmente, um sistema é composto por um conjunto de nós (CPUs), cada um gerido por um RTE. Em cada nó, várias tarefas são executadas concorrentemente.

A *tarefa* pode ser mapeada num dos três seguintes objectos estruturais SDL:

* **processo** (***TP - Task / Process mapping***): uma *tarefa* corresponde a um *processo* SDL. As *tarefas* são manipuladas pelo executivo e as *instâncias de processos* são manipuladas pelo GEODE dentro de cada *tarefa*. A implementação é descrita no ficheiro de configuração, no qual o utilizador associa as noções de *nó* e grupo de *tarefas* aos *blocos* (ou *sistema*) SDL da descrição. Estas associações têm de ser estabelecidas de acordo com a estrutura hierárquica da descrição.

* **bloco** ou **sistema** (***TB - Task / Block mapping***): uma *tarefa* corresponde a um *bloco* ou *sistema* SDL.

Estes mapeamentos são exemplificados na figura seguinte.

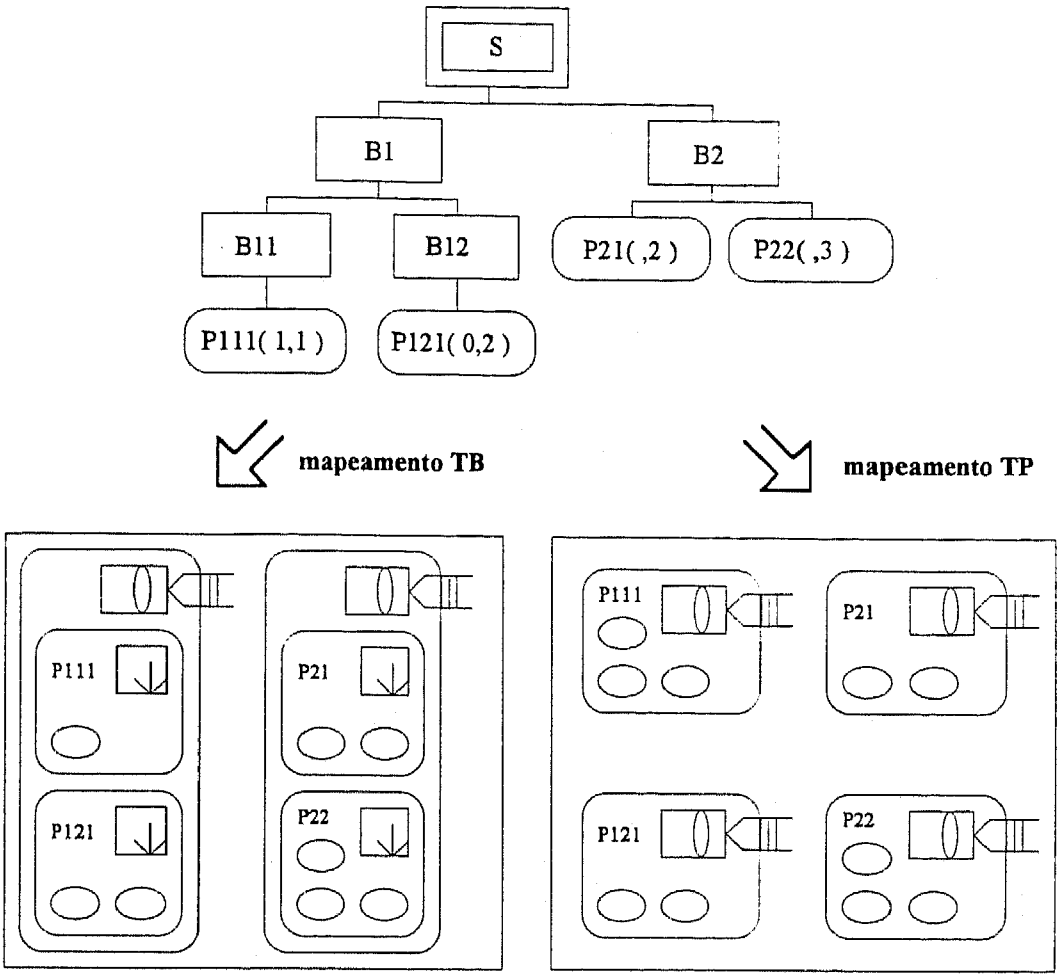


Fig. 3.8 - Mapeamentos entre a Arquitectura SDL e a Arquitectura Destino

Por omissão, *tarefas* RTE são mapeadas em *processos* (mapeamento TP); desta forma os diferentes processos correm num pseudo-paralelismo fornecido pelo RTE de acordo com as prioridades das tarefas definidas pelo utilizador. No caso do mapeamento TB todos os processos da mesma tarefa partilham a mesma fila e são executados sucessivamente de uma forma *event-driven*.

Máquinas de Estados

As máquinas de estados dos *processos* SDL são inteiramente traduzidas em linguagem C. O código gerado é completamente independente do executivo destino. O utilizador pode adaptar a aplicação integrando código e ficheiros de *include*.

No caso do mapeamento TP, a FSM de cada *processo* é implementada por uma função C, parecendo-se com uma tarefa típica RTE: espera por um novo *sinal* na fila; dependendo do estado corrente e do *sinal* recebido executa a sequência de transição, libertando depois o buffer e lendo o próximo sinal recebido, caso exista.

No caso do mapeamento TB, este *loop* é deslocado para o nível bloco e é composto por uma espera pelo novo *sinal* e pela chamada à função do *processo* correspondente.

ADTs (*Abstract Data Types*)

Por omissão, cada operador ADT descrito em SDL corresponde a uma função cuja interface C é gerada automaticamente. O corpo desta função deve ser escrito e a função compilada pelo utilizador. De cada vez que um operador é invocado num *processo* SDL, a função C correspondente é invocada no código gerado.

O utilizador pode pedir a geração de uma versão interactiva dos operadores seleccionados, com o propósito de testes. Neste caso o utilizador não tem de fornecer as funções destes operadores.

Tipos, constantes, variáveis e operadores declarados em SDL resultam, por omissão, na geração automática das correspondentes definições C. Contudo, o utilizador pode, por meio de uma opção especial no Gerador de Aplicação, introduzir numa descrição SDL as suas próprias equivalências entre declarações SDL e declarações C.

3.3.1.2 Ficheiro de Configuração

O Gerador de Aplicações GEODE usa o ficheiro de configuração (*descrição*.**cfg**), que descreve a arquitectura física da aplicação, além da sua descrição SDL.

Escrever o ficheiro de configuração é equivalente a associar as noções de *nó* com os *blocos* SDL (ou com o *sistema*), bem como *tarefas* e números prioritários com *processos* (mapeamento TP) ou *blocos* (mapeamento TB) SDL [VERILOG93c].

A sintaxe usada no ficheiro de configuração bem como um exemplo deste tipo de ficheiro são mostrados no Anexo B.

3.3.2 Geração do código fonte C da Aplicação

Durante esta etapa são gerados, na plataforma GEODE, os ficheiros de *include*, fonte e *makefile*.

No comando de geração (**geodeb_rte**) é especificado o Executivo em Tempo Real (*rte*) no qual a aplicação será executada e o nome da descrição (*descrição*) constituída pelos ficheiros de entrada (*descrição*.**cfg** e *descrição*.**gr**). Além destes dois parâmetros existe uma série de opções que podem ser consultadas no Anexo C.

O comando **geodeb_rte** começa por invocar o *script* **geodeb_*rte*_env**, podendo este ser configurado pelo utilizador para definir o ambiente de desenvolvimento (localização dos ficheiros, características dos ficheiros, tipo de mapeamento, etc.), executa de seguida uma série de invocações, e gera um conjunto de ficheiros [Fig. 3.9].

Como se pode ver na Fig. 3.9, é gerado um ficheiro C (*p_proc.c*) e uma tabela de estados indexados (*p_proc.h*) para cada *processo*, sendo estes os ficheiros correspondentes às FSMs da descrição SDL.

Os ficheiros *c_values.h c_declar.h* e *descrição.mk* são ficheiros de código C dependentes da configuração.

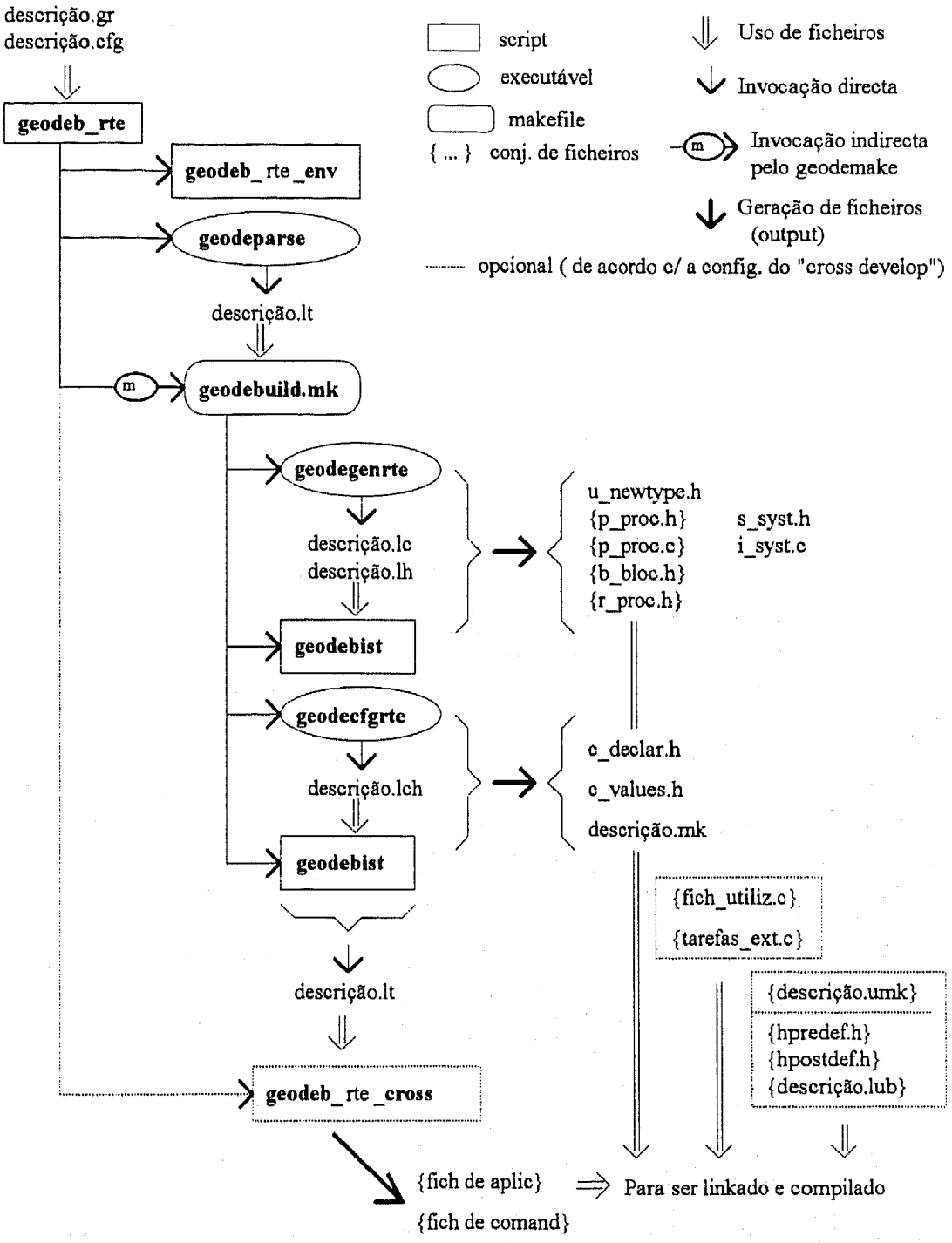


Fig. 3.9 - Sequência de Geração de Ficheiros

Operadores ADT

Para cada ADT com operadores, o gerador de código fornece um ficheiro específico (**u_*newtype*.h**) que define a interface das funções C a serem escritas pelo utilizador. O nome do ficheiro que contém estas funções é da escolha do utilizador (*fich_utiliz*.**c**) e deve ser compilado para se obter um ficheiro objecto (*fich_utiliz*.**o**). Os nomes destes ficheiros objecto devem ser adicionados ao ficheiro *descrição*.**lub**, usado para gerar os ficheiros executáveis da aplicação.

Se um operador ADT é declarado como sendo interactivo (usando as opções **-user** ou **-inter** descritas no Anexo C), qualquer código do utilizador para este ADT é ignorado.

Tipos C pré-definidos, constantes ou primitivas

Funções C, tipos e constantes que são pré-definidas ou pertencem a uma biblioteca podem ser integradas no código gerado.

As funções, tipos e constantes devem ser definidas na descrição SDL com um identificador que inclui um prefixo específico, num NEWTYPE, LITERAL, OPERATOR ou SYNTYPE.

A geração da aplicação deve então ser executada usando no comando de geração as opções **-prefix** *prefixo* ou **-fprefix** *ficheiro* (Anexo C). O código C gerado refere-se aos identificadores sem os seus prefixos e não fornece definições para estes identificadores. A definição deve ser feita no ficheiro **hpredef.h** ou deve ser implícita.

Consideremos o exemplo seguinte em que são apresentadas as definições contidas na descrição SDL. O código C gerado, usando o comando **geodeb_rte** *rte descrição* **-prefix** c_, irá fazer referência a *int*, *char* e *isdigit* sem dar a definição C.

EXEMPLO:

```
SYNTYPE c_int = INTEGER
     CONSTANTS -32765 : 32765
ENDSYNTYPE;

SYNTYPE c_char = CHARACTER
ENDSYNTYPE;

NEWTYPE c_functions
     OPERATORS
        c_isdigit : c_char -> c_int;
ENDNEWTYPE;
. . .
     DCL my_char c_char;
. . .
     TASK my_char := '3';
. . .
     DECISION c_isdigit (my_char);
        (0)  : . . .
        ELSE : . . .
```

Primitivas SDL

Todas as primitivas e definições SDL são traduzidas para C de acordo com regras específicas. Para aplicações concretas, o utilizador pode redefinir macros C para a realização de certas operações. Esta redefinição deve ser feita no ficheiro **hpostdef.h**, de acordo com a sintaxe das macros C, ou no ficheiro **h_*processo*.h** para um processo específico.

EXEMPLO:

SDL : `OUTPUT s to p;`

p_*processo*.c : `SDL_OUTPUT_s (p);`

p_*processo*.h :
```
#define     SDL_OUTPUT_s(pid) \
            G2S_CREATE_OUTPUT(sig_s, pid)
```

A redefinição pode ser a seguinte :

hpostdef.h :
```
#undef      SDL_OUTPUT_s
#define     SDL_OUTPUT_s(pid) \
            my_primitive(pid)
```

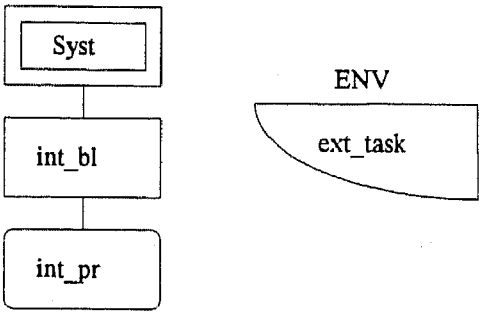
Tarefas Externas

O GEODE fornece facilidades para a integração de código escrito pelo utilizador para a implementação de partes da especificação SDL. Este tipo de código é definido como *tarefas* externas (*tarefas_ext*.c). No ficheiro de configuração, o utilizador pode declarar dois tipos de *tarefas* externas, para a implementação do exterior do sistema SDL e para a implementação de *blocos* ou *processos* SDL [VERILOG93c]. Para estas *tarefas*, o GEODE irá tratar da criação da *tarefa* RTE e de outros recursos, tais como filas, e fornecer os seus valores de PId.

Seguidamente são apresentados dois exemplos do uso de tarefas externas:

EXEMPLO 1: Tarefa externa para a implementação do exterior do sistema SDL

Diagrama Hierárquico:



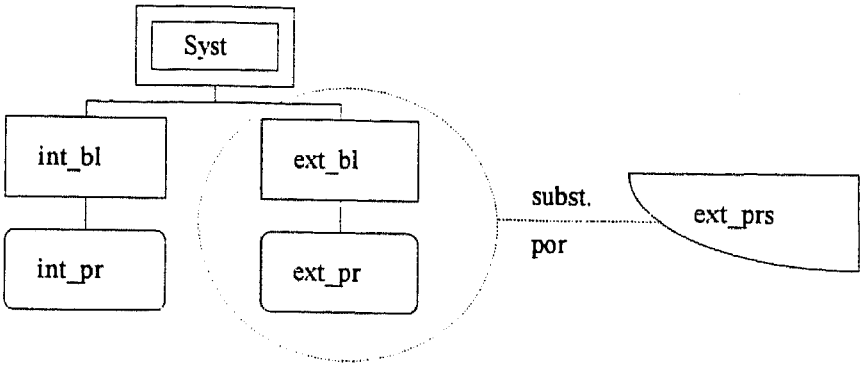
Configuração da Aplicação:

```
#BEGIN
#NODE syst
   #GROUP syst 1
                #EXTERN task -l ext_task
#END
```

A interpretação deste ficheiro de configuração, no que diz respeito à tarefa externa, pode ser feita da seguinte forma: a tarefa externa, correspondente à implementação do exterior do sistema SDL (`#EXTERN`), com o nome lógico *task* e com prioridade 1, encontra-se no ficheiro objecto *ext_task*.

EXEMPLO 2: Tarefa externa para a implementação de *processos* SDL

Diagrama Hierárquico:



Configuração da Aplicação:

```
#BEGIN
#NODE syst
   #GROUP syst 1
               #PROC int_pr 10
               #PROC ext_pr 10 ext_prs
#END
```

A interpretação deste ficheiro de configuração, no que diz respeito à tarefa externa, pode ser feita da seguinte forma: a tarefa externa, correspondente à implementação do processo SDL *ext_pr*, com prioridade 1, encontra-se no ficheiro objecto *ext_prs*. Como para o processo *int_pr* não é especificado nenhum nome (nem *path*) de ficheiro, este processo é um processo interno.

Capítulo 4

ANSAware

ANSA é uma arquitectura para ODP (*Open Distributed Processing*) que suporta o desenvolvimento e a construção de aplicações distribuídas independentemente da estrutura e do tamanho da rede, das características das máquinas ou dos Sistemas Operativos utilizados [ANSA93].

Uma das metas principais do ANSA é simplificar o projecto, a construção, o desenvolvimento e a gestão de aplicações distribuídas.

O ANSAware é um exemplo de implementação dessa arquitectura, sendo um pacote de software para construção de sistemas ODP e fornecendo uma plataforma básica de suporte para desenvolvimento de software, na forma de geradores de programas e aplicações de gestão de sistema.

Uma aplicação distribuída típica é construída a partir de alguns componentes cooperantes, sendo a cooperação entre pares de componentes dependente de uma definição precisa das interacções possíveis entre eles. O ANSAware fornece uma linguagem de especificação (IDL - *Interface Definition Language*) para definir o conjunto de interacções legais entre pares de componentes (especificações de interface) e uma ferramenta (*stubc - stub compiler*) para compilar essas especificações, dando origem a rotinas C (chamadas *stubs*) para serem incluídas nos programas que irão fornecer e usar essas interfaces.

O modelo de interacção suportado pelo ANSAware força todas as interacções entre componentes a um paradigma de invocação remota: o cliente de um serviço invoca uma operação, definida na especificação da interface, ao servidor que executa a operação posteriormente. Duas formas de invocação remota são suportadas: 1) *interrogation*, na qual o cliente que invoca a operação bloqueia até que o servidor retorne algum resultado, e 2) *announcement*, na qual o cliente não espera que o servidor execute a operação.

Uma vez efectuadas, em IDL, as especificações das interfaces de uma aplicação distribuída, os componentes que fornecem e usam essas interfaces devem ser codificados. Devido à diferença entre invocação remota e chamada a procedimentos locais, uma linguagem (PREPC) foi definida para invocar operações e aceder a outras facilidades ANSAware. Um pré-processador (prepc) é usado para converter programas fonte, em linguagem C com declarações PREPC, em ficheiros C standard.

4.1 Conceitos e Estrutura

Serviços

O bloco básico do ANSA é o **serviço**. Um serviço é uma função, que pode ser de processamento, de armazenamento ou de transferência. Os componentes que usam o serviço têm o nome de **clientes** e os componentes que fornecem o serviço são **servidores**. O modelo computacional ANSA permite que um componente seja simultaneamente cliente e servidor de vários serviços. Um componente ou objecto descrito puramente em termos da maneira como fornece e usa os serviços é referido como um objecto computacional.

Quando clientes usam o mesmo serviço, diz-se que partilham o mesmo serviço fornecido por servidores. O serviço é fornecido numa interface do servidor: uma interface é a unidade de fornecimento do serviço.

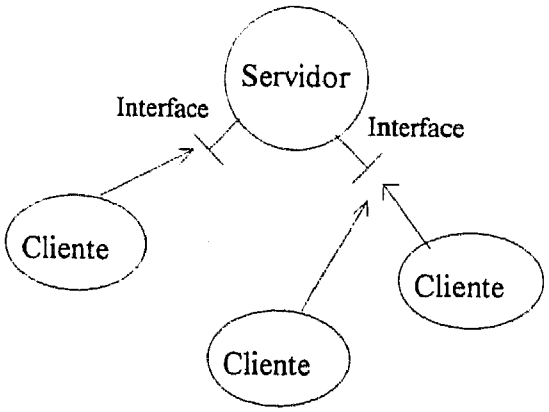


Fig. 4.1 - Servidores, Clientes e Interfaces

Os serviços podem ser divididos em **serviços de aplicação**, que são específicos das tarefas a serem fornecidas pelo sistema (por exemplo um serviço de reservas aéreas, um serviço multimédia, etc.), e **serviços de arquitectura**, que são genéricos para um conjunto de tarefas e, como o nome sugere, são fornecidos pela Arquitectura ANSA.

Os serviços de arquitectura fornecem meios consistentes para funções como encontrar serviços, controlo de acesso e gestão num sistema distribuído. Um exemplo dessas funções designa-se por ***trading*** e permite aos clientes encontrar dinamicamente servidores. O serviço arquitectural que fornece o *trading* é implementado no ANSAware por um objecto com o nome de ***Trader***.

Referências de Interfaces

Computacionalmente, os objectos comunicam entre si passando **referências de interfaces** uns aos outros. A posse de uma referência de interface por parte de um cliente permite a invocação das operações desse serviço fornecidas nessa interface pelo servidor. Estas referências podem ser passadas livremente de um objecto para outro, com ou sem um serviço intermediário de *trading*, isto é envolvendo ou não o ***trader***. O processo de fazer com que um serviço se torne público através de uma "oferta" deste ao *trader* é mostrado na figura seguinte.

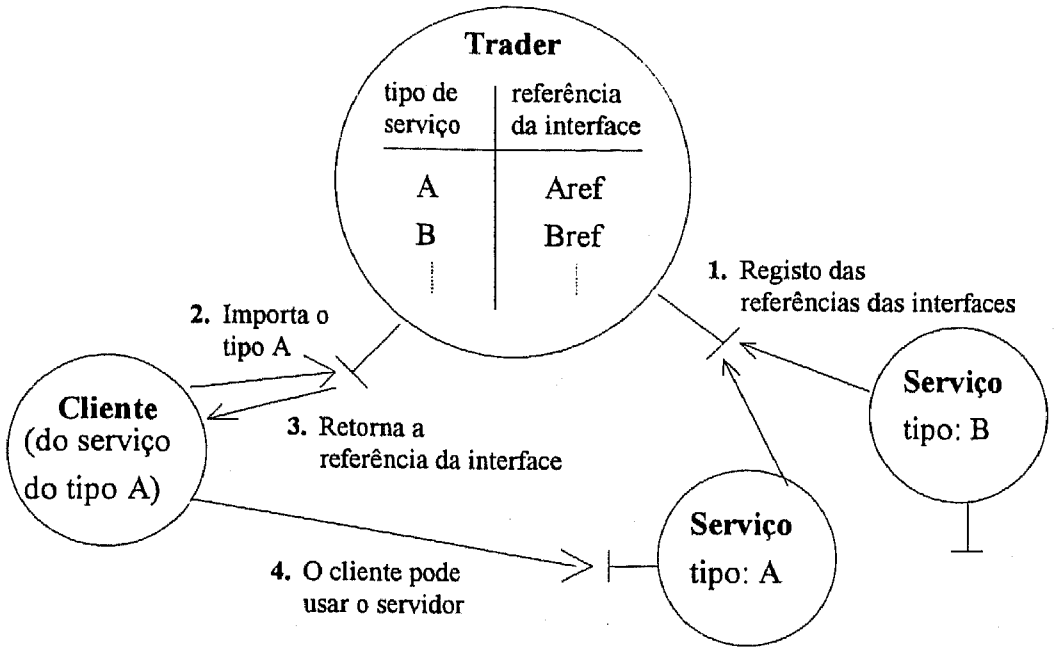


Fig. 4.2 - *Trading* Interfaces

Transparências

A transparência mascara os aspectos particulares da complexidade da programação distribuída. O ANSA identificou um número de transparências que podem ser individualmente activadas ou não, conforme a necessidade da aplicação.

As transparências que são implementadas pelo ANSAware são:

acesso - permite um estilo uniforme de interacção entre objectos, independente da sua construção, ambiente ou dos objectos serem locais ou remotos;

localização - permite interacção com um objecto sem necessidade de conhecer a sua localização fisica.

Existem mais transparências em discussão na Arquitectura ANSA que serão implementadas numa futura versão do ANSAware; uma delas é a **replicação**, que permite que múltiplas instâncias de uma interface sejam tratadas como uma, escondendo detalhes de organização de objectos que suportam este grupo de interfaces; outras transparências a serem suportadas são a **migração** (escondendo o movimento de objectos), **concorrência** e **atomicidade** (juntas fornecem suporte para transacções).

4.2 Ferramentas e Facilidades

4.2.1 IDL - *Interface Definition Language*

IDL é a linguagem de definição de interfaces que especifica as operações disponíveis numa interface. Todas as interacções entre objectos ANSAware são baseadas nas especificações das interfaces dos serviços e todos os detalhes que dizem respeito às interacções devem ser expressos em IDL.

A maior parte das linguagens de definição de interfaces até à data concentraram-se no aspecto sintáctico da interface. Estas linguagens fornecem um conjunto básico de primitivas de tipos de dados, de construções de tipos de dados mais complexos e um método para especificar operações. Exceptuando uma ou duas primitivas de tipos de dados e construções, os aspectos sintácticos da maior parte das IDLs existentes são similares.

No entanto, outros aspectos, além da sintaxe, são também importantes para facilitar as interacções entre os objectos, relacionados com informação semântica adicional que incide sobre:

- os tipos de interacções suportadas (*interrogations* ou *announcements*): estes modos de interacções dependem da escolha dos protocolos usados para suportar a interface;
- os requisitos de transparência das interacções.

O IDL do ANSAware representa esta informação explicitamente.

Tipos de Dados

IDL suporta vários tipos de dados e construtores de tipos de dados, sendo estes últimos usados para construir estruturas de dados mais complexas a partir dos tipos de dados básicos. Além disso, o ANSAware inclui um conjunto de tipos pré-construídos, que são usados para a comunicação entre serviços ANSAware normalizados, e que o ***stubc*** automaticamente disponibiliza às aplicações do utilizador. Desses tipos o mais frequentemente usado é o ansa_InterfaceRef.

Para cada um dos tipos são apresentadas no Anexo D uma descrição sumária, a declaração IDL e a declaração C correspondente.

Operações

Cada operação numa interface tem um nome, uma lista de argumentos e uma lista de resultados (podendo estas listas serem vazias). Os argumentos e os resultados são especificados pela posição e são passados por cópia de valor. Aos argumentos devem ser dados nomes de parâmetros formais.

Em ANSAware existem dois tipos de operações: *interrogation* e *announcement*. Se na definição da operação não se especifica de que tipo é, esta será interpretada como sendo do tipo *interrogation*. A lista de resultados de uma operação do tipo *announcement* deve ser sempre vazia.

Para ter uma ideia de como são definidas as operações, vamos considerar o seguinte exemplo que corresponde à definição de uma operação do tipo *interrogation*:

```
Nome_Op : INTERROGATION OPERATION [ arg1 : tp1; arg2 : tp2;
                                    . . .; argN : tpN ]
          RETURNS [ rtp1; . . .; tpK ];
```

Estrutura de uma Especificação

Uma especificação IDL tem a seguinte estrutura:

```
1 => Nome1 : INTERFACE =
2 => [IMPLEMENTATION] IS COMPATIBLE WITH Nome2 [FROM File];
3 => NEEDS Nome3 [FROM File];
     BEGIN
4 =>          -- Definição de contruções de tipos de dados
5 =>                    -- Definição de operações
     END.
```

Especificação IDL Genérica

onde:

1. O primeiro item da especificação é o nome do tipo de interface. Este nome é usado para referir esta interface noutras especificações nas

declarações IS COMPATIBLE WITH e NEEDS e pelo sistema *trading*.

2. Opcional. Declara que as instâncias *Nome1* também satisfazem a especificação *Nome2*. Todas as definições de tipos de dados e operações definidas em *Nome2* são adicionadas ao *Nome1*; só tipos de dados e operações adicionais são definidos nesta especificação. Esta compatibilidade é transitiva, isto é, se *Nome2* for compatível com *NomeX*, então *Nome1* também será compatível com *NomeX*. Pode existir na mesma especificação mais do que uma declaração IS COMPATIBLE WITH, que pode opcionalmente ser completada por FROM File se o nome do ficheiro IDL for usado como nome base para os ficheiros de saída em vez do nome da interface. Compatibilidade de interfaces permite um tipo seguro de ligação entre clientes e servidores. Se um cliente desejar usar uma instância de um tipo, por exemplo *Nome1*, a relação de compatibilidade permite que use uma instância de qualquer tipo que seja compatível com esse dado tipo, por exemplo *Nome2*. Se a declaração é da forma IMPLEMENTATION IS COMPATIBLE WITH, isso significa que da mesma forma que as definições dos tipos de dados e das operações são as mesmas que em *Nome2*, também o são as implementações das funções.
3. Opcional. Declara que a instância servidora de *Nome1* irá necessitar de aceder a instâncias de *Nome3*. Todos os tipos definidos em *Nome3* são adicionados a *Nome1*. Pode existir na mesma especificação mais do que uma declaração NEEDS. Isto torna-se mais útil quando referências a instâncias de *Nome3* são passadas a uma instância de *Nome1* como argumentos ou como resultados de uma operação. Como no item anteriormente referido, NEEDS pode opcionalmente ser seguido de FROM File pela mesma razão.
4. Opcional. Definição de tipos de dados.
5. Opcional. Definição de operações como descrito anteriormente em **Operações**.

4.2.2 stubc - *stub compiler*

Esta ferramenta do ANSAware compila especificações escritas em IDL em ficheiros *include* e ficheiros fonte cliente/servidor (*stubs*) em linguagem de programação C.

Para uma interface com, por exemplo, o nome **soma**, o **typedef somaRef** é automaticamente gerado pelo *stubc*. Este tipo **somaRef** pode então ser usado para fazer declarações nos ficheiros de implementação do servidor no PREPC, como por exemplo:

```
! DECLARE (f_ref) : soma  SERVER;
. . .
somaRef  f_ref;
```

O *stub* servidor necessita, para cada operação, de uma função C no código da implementação do programa servidor, função esta que é fornecida pelo utilizador. Por exemplo, consideremos a seguinte especificação:

```
Nome1 : INTERFACE =
BEGIN
     Operação1 : OPERATION [ arg1 : tp1 ; ... ; argM : tpM ]
                    RETURNS [ rtp1 ; ... ; rtpN ];
END.
```

A função C fornecida pelo utilizador deve ser declarada da seguinte forma:

```
int  Nome1_Operação1(_attr, arg1, ..., argM, res1, ..., resN)
     ansa_InterfaceAttr  *_attr;
     type1   arg1;
     . . .
     typeM   argM;
     rtype1   *res1;
     . . .
     rtypeN   *resN;
```

O valor de retorno desta função deve ser **SuccessfulInvocation** ou **UnsuccessfulInvocation**; o ponteiro **_attr** pode ser usado na rotina para aceder a um estado específico da instância.

Se a operação for completada com sucesso, deve retornar **SuccessfulInvocation**; e, além disso, o *stub* servidor deve colocar os resultados em **res1 ... resN** para serem retornados a quem os solicitou. Se for retornado pela função o valor **UnsuccessfulInvocation**, o *stub* servidor não colocará os resultados; em vez disso é retornado ao cliente **abnormalReturn**. Isto é útil caso ocorra um erro enquanto é processado o pedido, que impossibilite a geração dos dados.

Especificado a *flag* **-t** na linha de comando para o compilador *stubc* faz com que este gere um ficheiro amostra que contém as declarações correctas para as operações na especificação. Cada rotina neste ficheiro não faz nada e retorna **UnsuccessfulInvocation**; editando este ficheiro e completando a implementação de cada operação, o programador pode desenvolver o código da aplicação.

Compatibilidade de Interfaces

Foi explicado anteriormente que uma especificação de uma interface com nome **Nome1** e uma operação **Operação1** irá requerer a função **Nome1_Operação1** no código que implementa a interface. Havendo compatibilidade de interfaces (usando a declaração **IS COMPATIBLE WITH**) vão existir algumas diferenças ligeiras nos nomes das funções que são fornecidas pela implementação da interface.

Considerando o seguinte exemplo de uma interface do tipo **CalcOps**, a interface é definida como tendo uma operação **Soma**. Neste exemplo, a operação **Soma** aceita dois argumentos do tipo **REAL**, e retornam um resultado também do tipo **REAL**. A especificação da interface é então:

```
CalcOps : INTERFACE =
BEGIN
     Soma : OPERATION [ N1, N2 : REAL ]  RETURNS [ REAL ];
                          RETURNS [ REAL ];
END.
```

Isto significa que o ficheiro **.c** ou **.dpl** que implementa a operação descrita nesta interface deve ter a função:

CalcOps_Soma ( )

O programador pode agora desejar definir outra interface, do tipo **MaisCalcOps**, que fornece a operação que a **CalcOps** fornece mais uma operação

adicional **Inverso**. Para efectuar isto, existem duas opções baseadas na compatibilidade de interface.

* Se a nova implementação da interface é para tornar a implementar a operação da interface **CalcOps**, então a definição pode ser feita da seguinte forma:

```
MaisCalcOps : INTERFACE =
IS COMPATIBLE WITH CalcOps;
BEGIN
     Inverso : OPERATION [ N : REAL ]  RETURNS [ REAL ];
END.
```

Usando esta opção, ANSAware espera que o código que implementa a especificação **MaisCalcOps** contenha implementações de todas as operações especificadas na interface, isto é:

MaisCalcOps_Soma ( )
MaisCalcOps_Inverso ( ).

Isto significa que, embora a interface do tipo **MaisCalcOps** esteja em conformidade com a interface do tipo **CalcOps**, pode ter a sua própria possibilidade de implementação, ligeiramente diferente, das operações definidas.

* A segunda opção é usar a implementação do servidor da operação de **CalcOps**. Sendo a definição da interface feita da seguinte forma:

```
MaisCalcOps : INTERFACE =
IMPLEMENTATION IS COMPATIBLE WITH CalcOps;
BEGIN
     Inverso : OPERATION [ N : REAL ] RETURNS [ REAL ];
END.
```

Usando esta definição, as ferramentas ANSAware esperam que o código que implementa a especificação **MaisCalcOps** contenha a operação:

MaisCalcOps_Inverso ( )

e o código para a operação CalcOps_Soma ( ) será também linkado. Desta forma, o código existente para a implementação de CalcOps pode ser usado, se tal for desejável.

É claro que é o programador que estabelece as regras correctas de linkagem no **Imakefile** ou **Makefile** para contemplar estas possibilidades.

4.2.3 *Trader*

Uma referência de interface pode ser passada de componente para componente como um argumento de uma invocação de uma operação ou como resultado de uma invocação. O componente que recebe a referência de interface pode imediatamente começar a invocar operações na interface.

Um aspecto importante da construção de aplicações distribuídas é a ligação dinâmica a serviços existentes. O *Trader* é usado por servidores para anunciar serviços com os atributos apropriados e é usado por clientes para localizar os serviços oferecidos [Herbert94]. Este processo é denominado *trading* [Fig. 4.3].

Os servidores registam as instâncias de interfaces no *Trader* para que os clientes as possam localizar. O registo de uma instância de interface no *Trader* é referido como uma **oferta**. Diz-se que o servidor oferece o(s) serviço(s) para uso de potenciais clientes.

Um cliente, que deseje localizar um dado tipo de serviço, deve fazer uma invocação ao *Trader*, para obter a referência de interface desejada. Depois do cliente possuir a referência de interface do servidor pode invocar qualquer operação definida nessa interface, devendo libertar os recursos após cada invocação.

Da mesma forma que o servidor regista as instâncias de interface no *Trader*, é livre de as remover deste.

Um problema a ter em conta é como um cliente escolhe uma instância de interface de um conjunto de instâncias que suportam o mesmo tipo de interface. Para ajudar esta escolha, o ANSAware permite associar propriedades às instâncias. Assim, quando uma instância de interface é registada no *Trader* podem-se associar a essa instância pares **(Nome_Propriedade, valor)**. Quando um cliente importa do *Trader* uma referência de interface deve indicar as propriedades e os respectivos valores que a instância deve possuir (restrições).

Os nomes das propriedades são caracteres alfanuméricos, sendo o primeiro caracter uma letra. Os valores podem ser números, palavras ou um conjunto de palavras. Devendo neste último caso, as palavras estar dentro de {}.

O ANSAware permite ainda que o cliente especifique restrições complexas usando lógica **and, or e not** entre os valores das diferentes propriedades.

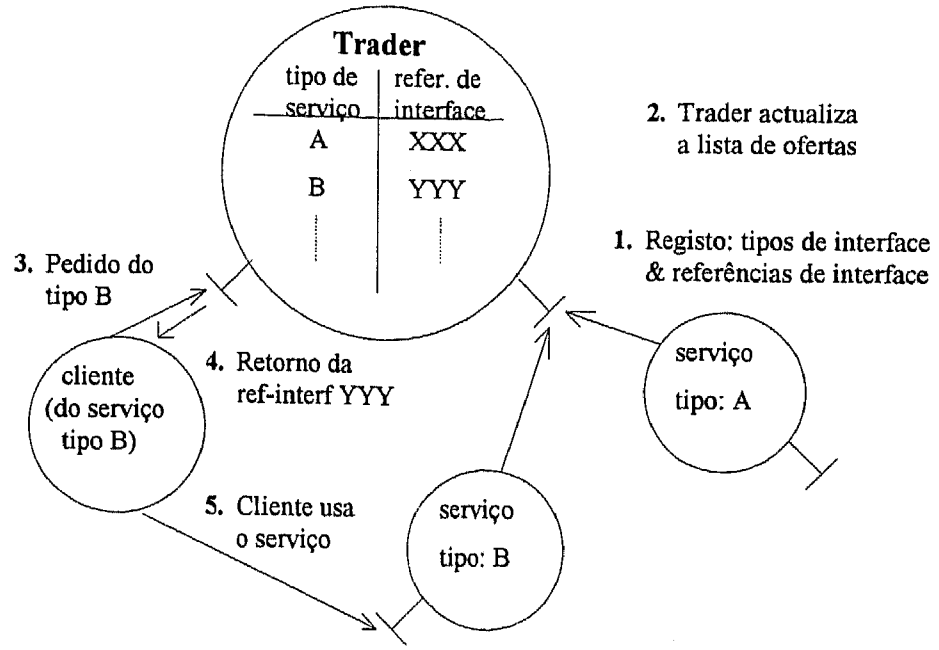


Fig. 4.3 - *Trading*

4.2.4 Cápsulas, Objectos e Interfaces

A cápsula é a unidade de operação autónoma em ANSAware e uma interface é o ponto de fornecimento de serviço. Cápsulas, objectos, interfaces e operações são os níveis de encapsulamento em ANSAware, e as relações entre eles podem ser descritas da seguinte forma:

Cápsula: conjunto de 0 ou mais objectos,
Objecto: conjunto de 0 ou mais interfaces,
Interface: conjunto de 0 ou mais operações,
Operação: especificação de tipos de argumentos e resultados,
em conjunto com uma implementação.

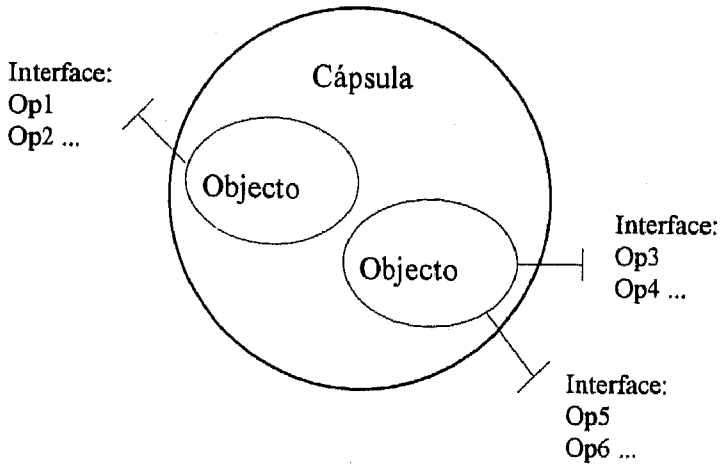


Fig. 4.4 - Cápsulas, Objectos e Interfaces

O Objecto é a unidade usada pela ***Factory*** para instânciação dinâmica, como é explicado na secção seguinte.

4.2.5 *Factory*

A maior parte dos sistemas orientados a objectos, e o ANSA não é excepção, suportam alguma noção de instânciação dinâmica de objectos, sendo o local onde os objectos são criados designado por *Factory* (Fábrica) Nesta secção é descrito o suporte ANSAware para instânciação dinâmica de objectos, sendo mostrados na Fig. 4.5 os passos necessários para a criação de um objecto.

A instanciação dinâmica de um objecto é realizada instanciando uma cápsula que contém um objecto com a interface desejada. Esta cápsula é encarada como uma amostra, pois quando é iniciada pela *Factory* não são instanciados nenhuns objectos nem interfaces dentro dela; ela é apenas uma amostra com a capacidade de suportar vários objectos e interfaces (amostras) que poderão mais tarde ser instanciados.

(1) O cliente pede à *Factory* para instanciar uma cápsula.

(2) A *Factory* cria uma nova cápsula.

(3) A nova cápsula contém um só objecto com uma interface do tipo *cápsula*. Este objecto é um criador de objectos. A interface do tipo *cápsula* suporta a operação **CapsuleSInstantiate** que é usada para instanciar objectos nessa cápsula.

(4) A referência da interface *cápsula* é retornda à *Factory*.

(5) Que por sua vez a retorna ao cliente.

(6) O cliente pode então usar a referência da interface *cápsula* para criar os objectos de que necessita.

(7) Depois de receber esta invocação, o objecto criador de objectos da nova cápsula cria um novo objecto, e retorna um conjunto de referências de instâncias de interfaces do novo objecto para o cliente.

A operação **Capsule$Instantiate** (disponibilizada pela arquitectura ANSA) decide qual o objecto a criar, uma vez que existe mais do que uma possibilidade, com base no nome do objecto passado por quem invoca.

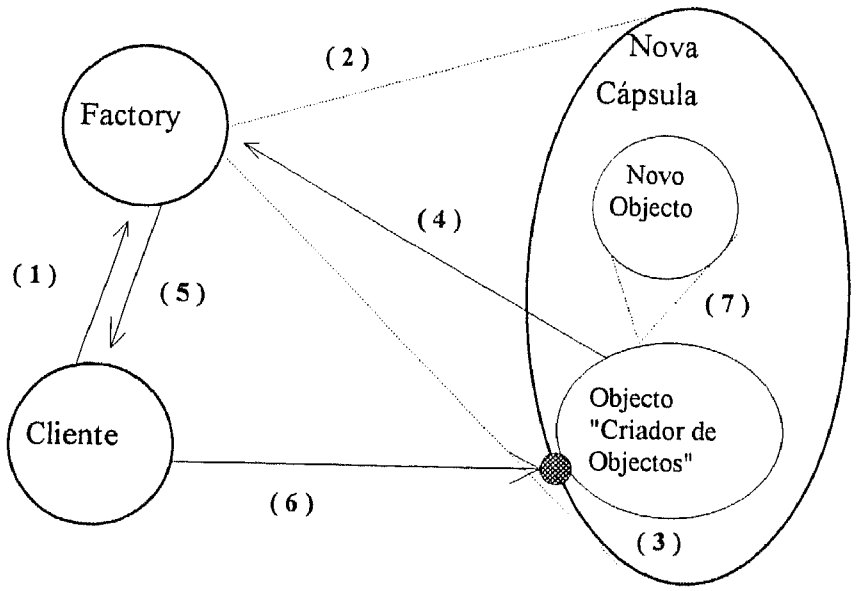


Fig. 4.5 - Interacções com a *Factory*

4.2.6 Linguagem PREPC

O código de programa para usar ou fornecer um serviço em ANSAware é escrito usando declarações PREPC e código C. Estas declarações suportam três facilidades básicas:

- Declarações de tipos de interface e ligação léxica de variáveis **ansa_InterfaceRef**;
- Criação/Destruição dinâmica de instâncias de interfaces;
- Invocação de operações numa ou mais instâncias de interfaces.

Declarações de Interfaces e Ligações Léxicas

Cada ficheiro fonte deve declarar os nomes de todos os tipos de interfaces que serão referenciadas no ficheiro, o que é feito com declarações da forma:

```
!  USE Nome1
!  USE Nome1 FROM File
```

Se for especificado o **FROM**, o PREPC irá procurar a especificação da interface **Nome1** no ficheiro **File** (.idl), em vez de no ficheiro **Nome1.idl**. Depois de uma declaração destas, todos os tipos de dados definidos na especificação da interface estão disponíveis para serem usados no ficheiro fonte.

O programador deve também relacionar variáveis **ansa_InterfaceRef** no programa com os tipos de interfaces com as quais estão ligadas. Isto é feito com declarações da forma:

```
!  DECLARE { Variáveis } : Nome CLIENT | SERVER
```

no qual:

Variáveis

Lista de variáveis **ansa_InterfaceRef** declaradas no ficheiro fonte. Deve-se ter em conta que uma declaração **! DECLARE** não elimina a necessidade de declarar as variáveis, pois simplesmente estabelece uma correspondência entre tipos IDL e tipos C;

Nome

O nome do tipo de interface para ser associada a estas variáveis;

CLIENT | SERVER

Indica que regra o programa irá assumir no que diz respeito a esta referência de interface; **CLIENT** indica que o programa irá fazer pedidos na interface, **SERVER** indica que o programa irá fornecer o serviço na interface.

A declaração C de variáveis do tipo **ansa_InterfaceRef** continua a ser necessária para variáveis especificadas com ! **DECLARE**. Isto significa que se a variável **var1** é declarada como se segue:

```
! DECLARE ( var1 ) : InterfNome SERVER
```

então a variável **var1**, antes de ser usada em qualquer parte do programa, deve ser declarada da seguinte forma:

```
ansa_InterfaceRef  var1;   ou   InterfNomeRef    var1;
```

É possivél usar **InterfNomeRef** em vez de **ansa_InterfaceRef**, porque o *stubc* gera automaticamente o tipo **InterfNomeRef** (que é equivalente a **ansa_InterfaceRef**) para uma interface com o nome **InterfNome**.

Criação/Destruição dinâmica de instâncias de interfaces

Fornecedores de serviços podem criar múltiplas instâncias de um tipo particular de interface, o que é conseguido através de declarações que o PREPC fornece para criar e destruir instâncias de interfaces.

Para criar uma instância de um tipo particular de interface deve-se usar:

```
! {variável} :: Nome$Create(concorrência[, ArgsOpcionais]),
```

na qual:

variável

Deve ser uma variável **ansa_InterfaceRef** previamente mencionada em ! **DECLARE**;

Nome

O nome do tipo de interface;

concorrência

Valor inteiro que indica o número máximo de pedidos que podem ser processados concorrentemente pela instância em qualquer instante;

ArgsOpcionais

Se os argumentos opcionais forem especificados, então o utilizador deve fornecer funções para criar e destruir a instância específica.

Para destruir uma instância de um tipo particular de interface deve-se usar:

```
!  ( ) :: Nome $ Destroy(variável)
```

com:

Nome

O mesmo que para **Create**;

variável

Uma variável **ansa_InterfaceRef** previamente associada com uma interface instanciada.

Invocação de Operações

A sintaxe genérica para uma invocação de uma operação é:

```
!  ( resultados ) <- ifref$op ( argumentos ) [ excepções ]
```

na qual:

resultados

É uma lista (possivelmente vazia) de resultados a serem retornados;

ifref

Variável **ansa_InterfaceRef**; esta referência ou é já conhecida ou então é recebida como argumento ou resultado duma invocação;

op

Nome da operação a invocar naquela interface;

argumentos

Lista (possivelmente vazia) de argumentos;

excepções

Opcional: uma declaração de acções a ter em conta quando se verificam as excepções especificadas.

Quando é encontrada uma invocação de uma operação, os **argumentos** são colocados num *buffer*, pelas rotinas do *stub* cliente, e enviados ao servidor correspondente à **ifref**. Se **op** for uma operação do tipo *interrogation*, a actividade de quem invoca é bloqueada até que o servidor tenha processado o pedido e retornado os **resultados**; se **op** for do tipo *announcement*, a actividade é retomada imediatamente.

Todos os pedidos de uma actividade particular a uma dada instância de interface são colocados em série pela infraestrutura. Pedidos de diferentes actividades clientes à mesma instância podem ser executados por qualquer ordem. Este facto pode ser explorado pelo programador, podendo ser criadas múltiplas actividades, cada uma executando uma interrogação em paralelo.

O PREPC fornece sintaxes para iniciar uma interrogação e subsequentemente recolher os resultados: iniciar uma operação de interrogação (usando ": ="), em vez de a invocar (usando "<-", como descrito anteriormente), não bloqueia a actividade; a execução continua depois da instrução de iniciar, e só bloqueará se os resultados ainda não estiverem disponíveis quando for feita a sua recolha (usando a função **"Redeem"**). A sintaxe para iniciar operações e recolher resultados é a seguinte:

```
ansa_Voucher  v;

/* Iniciar o pedido */
!  (v) := ifref$op( argumentos )

/* processamento feito em paralelo com a invocação, incluindo */
/* outras iniciações de operações na mesma interface */

/* recolha de resultados */
!  (resultados) <- ifref$Redeem(v) excepções
```

4.2.7 Pré-processador - prepc

O pré-processador fornecido com o ANSAware processa ficheiros C com declarações PREPC transformando-os em ficheiros fonte C. Consideremos um ficheiro

com nome **exemp.dpl** que contém C e declarações PREPC. A aplicação do pré-processador a este ficheiro dá origem aos seguintes ficheiros na directoria corrente:

exemp.c

Ficheiro fonte C com todas as declarações PREPC traduzidas em chamadas apropriadas à biblioteca *run-time* e a rotinas *stub*.

exemp.ah

Ficheiro que é incluído em **exemp.c** com um **#include**.

O pré-processador localiza os ficheiros relevantes para os tipos de interfaces mencionado na directiva "**! USE**". Por omissão estes ficheiros são procurados na directoria corrente. O programador pode especificar outras directorias adicionais usando a *flag* **-I**.

Três operações adicionais disponíveis nas instâncias do tipo ***Trader*** são construídas através do pré-processador. Estas operações são:

```
Import : OPERATION [ type : STRING; context : STRING;
                     constraints : STRING ]
     RETURNS [ InterfaceRef ];

Export : OPERATION [ type : STRING; context : STRING;
                     constraints : STRING ]
     RETURNS [ InterfaceRef ];

Withdraw : OPERATION [ ref : InterfaceRef ]
     RETURNS [ ];
```

A primeira permite aos clientes importarem do *Trader* as referências de interfaces pretendidas, a segunda permite aos servidores registarem no *Trader* as suas referências de interfaces e finalmente a última permite que os servidores retirem do *Trader* referências de interfaces.

Uma operação adicional é definida para todos os tipos, e tem a seguinte especificação:

```
Discard : OPERATION [ ]  RETURNS [ ];
```

A invocação desta operação notifica o ambiente de suporte de que deve terminar a ligação corrente ao serviço representado pela referência de interface, libertando os recursos que já não são necessários.

Nenhuma destas operações é essencial para a construção correcta de programas distribuídos.

4.3 Um Exemplo em PREPC

Considerando uma versão simplificada do exemplo usado na secção 4.3.2, no qual se pretende implementar um serviço do tipo **CalcOps** que contém uma só operação **Soma**, a sua especificação IDL pode ser a seguinte:

```
CalcOps : INTERFACE =
BEGIN
     Soma : OPERATION [ N1, N2 : REAL ]  RETURNS [ REAL ];
END.
```

4.3.1 Servidor.dpl

O código do servidor deve conter um **body( )**, rotina para criar uma instância da interface **CalcOps** e para a registar no *Trader*, e a implementação da operação **Soma**. Isto pode ser feito da seguinte forma:

```
!  USE CalcOps
!  USE Trader
!  DECLARE { ref } : CalcOps SERVER

/* body do serviço - simplesmente cria a instancia */
/* e exporta a oferta */

void body ( int argc, char *argv[ ], char *envp[ ] )
{
     ansa_InterfaceRef    ref;
!    { ref } :: Soma$Create( 16 )
!    { } <- traderRef$Export ( "CalcOps", "/ansa/exemp", " ",
ref )
}
```

A operação **Soma** irá retornar a soma dos dois números que recebeu. Observando a especificação IDL e seguindo as regras para a declaração de funções

(secção 4.3.2), esta função deve ter três argumentos, todos do tipo **ansa_Real**. Então a implementação pode ser a seguinte:

```
int CalcOps_Soma ( ansa_InterfaceAttr *_attr, ansa_Real n,
              ansa_Real m, ansa_Real   *result )
{
     *result = n+m;
     return SuccessfulInvocation;
}
```

4.3.2 Cliente.dpl

O programa cliente para ter acesso a este serviço tem de importar primeiro uma interface do tipo **CalcOps** do *trader* e, de seguida, invocar a operação **Soma** da interface **CalcOps**. No código exemplo mostrado, o programa cliente obtém os números (argumentos da operação **Soma**) a partir de **stdin**, colocando ">" no ecran para que o utilizador introduza os números.

```
ansa_Real a, b, c;

!  USE CalcOps
!  DECLARE { Ref } : CalcOps CLIENT

void body ( int argc, char *argv[ ], char *envp[ ] )
{
     ansa_InterfaceRef    Ref;
!    { Ref } <- traderRef$Import( "CalcOps", "/", " " )
     printf(">");
     scanf("%f",&a);
     printf(">");
     scanf("%f",&b);
!    { c } <- Ref$Soma( a, b )
     printf("%f", c);
!    Ref$Discard
}
```

Capítulo 5

Tradução SDL-ANSA

Sendo o SDL uma linguagem que permite descrever e especificar serviços de uma forma rápida e o ANSA uma arquitectura para ODP que suporta o desenvolvimento e construção de aplicações distribuídas, a tradução de SDL para ANSA torna-se bastante útil para a criação rápida de serviços em ambientes distribuídos.

5.1 Correspondência SDL-ANSA

O SDL (abreviatura que será sempre usada para SDL-88) é uma linguagem *baseada em processos* que comunicam entre si por meio de mensagens. Esta linguagem e o modelo computacional ANSAware (*orientado a objectos*) apresentam algumas características semelhantes, como por exemplo a interacção entre componentes através de mensagens. No entanto, ainda existem diferenças significativas que tornam impossível a correspondência directa entre o SDL e a plataforma ANSAware.

5.1.1 Alternativas para a Correspondência

Existem duas alternativas para transformar uma especificação SDL numa implementação ANSAware.

A primeira é fazer a correspondência entre os conceitos dos dois modelos sem entrar em consideração com a forma como estes são utilizados. Desta forma, esta alternativa depende da proximidade dos conceitos SDL e ANSAware.

Um conceito chave em SDL é o processo. Sendo este a unidade dinâmica, aproxima-se muito do conceito de objecto em ANSAware. Outro conceito importante, em ANSAware, é a interface, a unidade de referência. O SDL carece de um conceito similar, o que implica que as interfaces devam ser emuladas em SDL. Emular interfaces não está no contexto desta alternativa, pois estaríamos a considerar o uso individual deste conceito. Da mesma forma, conceitos SDL que não existem em ANSAware causarão emulação extensiva e irão forçar os objectos ANSAware a conter suporte *run-time* SDL (ex: a semântica *save*).

A segunda alternativa consiste em modelar os conceitos ANSAware em SDL, o que significa que um projecto SDL terá de ser expresso segundo os conceitos ANSAware. Desta forma, um conjunto de regras deve ser aplicada ao uso do SDL para uma aproximação entre os conceitos dos dois modelos.

Existem duas possibilidades para esta alternativa: i) fazer corresponder os blocos a objectos ou ii) fazer corresponder os processos a objectos.

Blocos como Objectos

Como os blocos não têm uma representação *run-time* nem suportam encapsulação de dados, deve ser adicionado na especificação SDL um processo especial (monitor) para encapsulação, dentro do bloco, de dados e procedimentos do objecto.

A correspondência baseada na unidade de concorrência irá fazer corresponder os processos SDL a *threads* ANSAware. Isto é, deve ser criado um processo SDL de cada vez que uma operação é invocada nesse bloco. Isto leva a que deva existir um processo especial para criar os processos, podendo este processo ser o monitor (referido anteriormente). Da mesma forma, a correspondência baseada na unidade de referência leva a que a processos SDL correspondam interfaces ANSAware. Isto é, como os processos suportam operações diferentes uns dos outros, a referência a uma interface, que fornece uma dada operação, deve corresponder à referência do respectivo processo.

Desta forma, a maior parte dos conceitos ANSAware estarão cobertos, mas como o processo (conceito SDL) é usado para modelar alguns conceitos ANSAware, esta possibilidade leva-nos a uma emulação pesada, o que não é de grande interesse.

Processos como Objectos

A correspondência entre processos SDL e objectos ANSAware faz com que a maior parte dos conceitos ANSAware possam ser modelados, isto é, as interfaces são modeladas por vias (que interligam processos SDL) e as operações por sinais.

Desta forma não são necessárias as emulações referidas anteriormente. A desvantagem principal é que alguns conceitos ANSAware são modelados de uma forma restrita, como por exemplo, uma interface não pode ser modelada como uma entidade de referência.

5.1.2 Correspondência Usada

As duas alternativas apresentadas anteriormente possuem vantagens e desvantagens. A primeira oferece liberdade de expressão no modelo fonte, mas leva a que conceitos importantes do modelo destino não sejam usados. A segunda alternativa assegura que o modelo destino é bem explorado, mas pode causar emulação pesada e expressões incómodas no modelo fonte.

A escolha da alternativa depende do que se pretende fazer. Uma vez que com o trabalho realizado se pretendia a criação rápida de serviços de telecomunicações, especificando-os em SDL e traduzindo-os para que fossem suportados pela plataforma ANSAware, pode-se fazer a seguinte observação:

É importante explorar os conceitos ANSAware, mas por outro lado um projecto SDL baseado na emulação dos conceitos destino está fora dos objectivos, uma vez que assim estaríamos a especificar um serviço suportado pelo ANSAware em SDL, o que contraria o objectivo do SDL como linguagem de especificação. Uma especificação SDL deve ser a representação do problema e não deve ser afectada por emulações de conceitos da infraestrutura destino. O importante é encontrar uma forma de correspondência que minimize a emulação e maximize o número de conceitos ANSAware com correspondência SDL.

Correspondência

A tabela seguinte mostra a correspondência estabelecida entre conceitos SDL e conceitos ANSAware.

SDL	ANSAware
Tipo de Processo	Amostra de Objecto
Instância de Processo	Objecto
Via	Amostra de Interface
PId	Lista de Referências de Interfaces
Sinal	Operação *announcement*
Saída de um Sinal	Chamada a uma Operação *announcement*
Entrada de um Sinal	Activação de uma Operação *announcement*

Tabela 5.1 - Correspondência usada

Cada instância de um processo (objecto) é capaz de actuar como servidora para as interfaces a ela associadas.

Os sinais de entrada mencionados numa via correspondem a operações que as interfaces fornecem, sendo estas operações de *announcement*. As operações de *interrogation* não têm uma correspondência em SDL, pois a comunicação em SDL é assíncrona. Se a versão do SDL suportasse chamadas a procedimentos remotos, estes sim corresponderiam a operações de *interrogation* e consequentemente as chamadas a estes procedimentos corresponderiam à activação de operações de *interrogation*.

A Fig. 5.1 mostra um exemplo da correspondência entre uma especificação SDL e uma implementação ANSAware. As vias cuja direcção é a de entrada no processo correspondem a interfaces. A instância do processo fornece as operações s1, s2 e s3 na via Sr1, e s4 e s5 na via Sr2, o que corresponde em ANSAware ao fornecimento dessas operações nas interfaces respectivas.

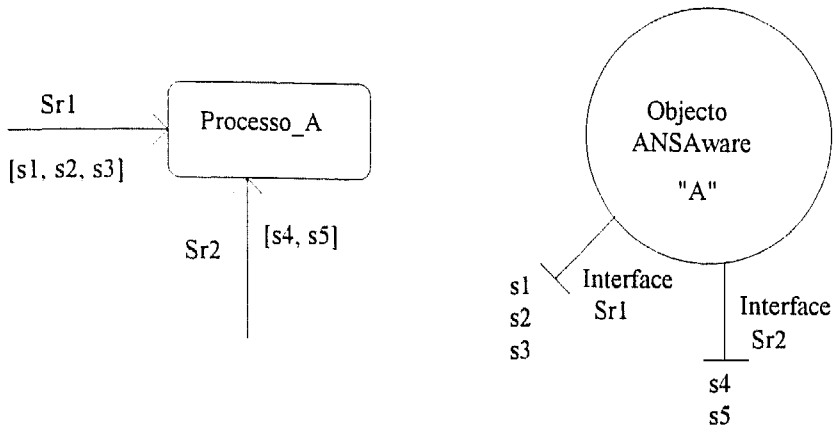


Fig. 5.1 - Correspondência SDL ↔ ANSAware:
Processo só com sinais de entrada

Uma via cuja direcção é a de saída de um processo não corresponde a nenhum conceito ANSAware, indicando simplesmente que este processo SDL (ou objecto ANSAware) é cliente de outro. Um exemplo desta situação é mostrado na Fig. 5.2. O objecto A é servidor das operações s1, s2 e s3 na interface Sr1 e das operações s4 e s5 na interface Sr21; simultaneamente este objecto é cliente do objecto B que fornece as operações s6 e s7 na interface Sr22. Isto significa que uma via bidireccional que interliga dois processos SDL corresponde a duas interfaces ANSAware.

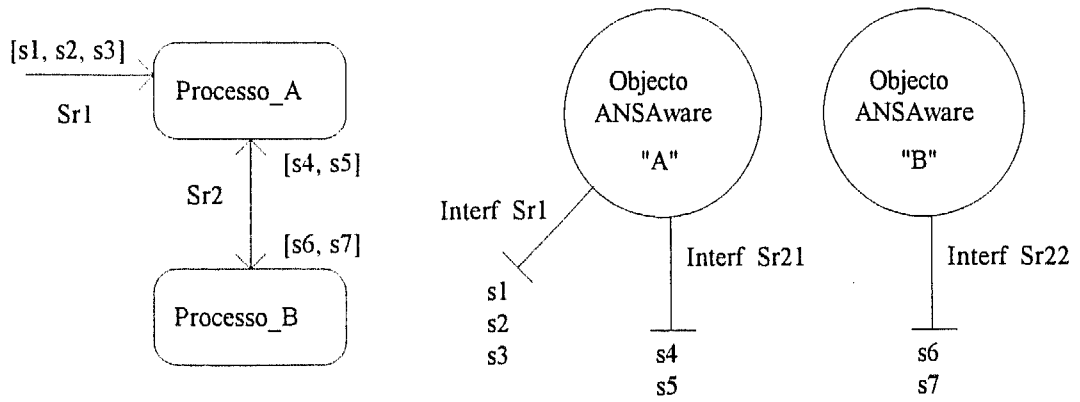


Fig. 5.2 - Correspondência SDL ↔ ANSAware:
Processo com sinais de entrada e de saída

O ANSAware requer que cada operação fornecida por um dado objecto seja fornecida unicamente numa interface desse objecto, o que leva a que um sinal de entrada num processo só possa ocorrer numa via desse processo. A via a ser usada pode ser sempre identificada se se conhecer o sinal a ser enviado e a instância do processo destino, identificada pelo valor PId. Como o PId do processo é único qualquer que seja a via a ser usada, o conceito correspondente em ANSAware é a lista de referências de interfaces, uma para cada via de entrada do processo tipo.

Isto é, em SDL, a instância do processo **M**, que cria uma instância de um processo **servidor**, pode passar o PId do servidor a uma terceira instância de um processo **cliente**, que pode então enviar sinais ao servidor. Em geral M não sabe qual a via do servidor que o cliente irá usar. Mas se o cliente tiver as referências de todas as interfaces do servidor, o cliente pode escolher a referência da interface apropriada para o sinal que quer enviar.

Apesar das observações feitas e, por motivo de limitação de tempo, o tipo PId não foi explorado em pormenor, podendo ser uma área para pesquisa futura. No entanto, como referido anteriormente (secção 4.2.5), o ANSAware suporta instanciação dinâmica de objectos através do uso da *Factory*. O que leva a que a instância do processo **M**, criadora de processos **servidores**, irá corresponder ao objecto criador de objectos, em ANSAware [Fig. 4.5], e irá retornar ao cliente um conjunto de referências de interfaces do servidor.

Trading

Na especificação SDL a noção de *Trader* não será visível. Esta noção será introduzida durante a tradução para a plataforma ANSAware, o que implica que o uso explícito das funcionalidades do *Trader* deva ser descrito como parte da especificação SDL, como, por exemplo, obter o PId do processo servidor. Mas como numa especificação SDL normal é desta forma que se efectua o manuseamento do PId, a obtenção do PId do processo servidor não é um problema.

Uma alternativa seria descrever todo o *Trader* ANSAware na especificação SDL. Mas esta alternativa iria requerer uma emulação pesada, o que não é pretendido.

5.2 Tradução SDL-88 para ANSAware

Tendo definido a correspondência entre os conceitos SDL e ANSA, o passo seguinte consiste na tradução dos ficheiros gerados pela plataforma GEODE, a partir de uma especificação SDL, em ficheiros suportados pela plataforma ANSAware.

5.2.1 Ficheiros utilizados

O ANSAware requer descrições separadas das interfaces e da parte operacional do objecto. Esta última é obtida a partir dos ficheiros **.h** e **.c** de cada processo existentes na especificação e fornecidos pelo Gerador de Aplicações GEODE (secção 3.3.2). Como os ficheiros produzidos por este Gerador não fazem qualquer referência às vias existentes na especificação, as descrições das interfaces são obtidas a partir do ficheiro *descrição*.**pr**.

O processo de tradução é mostrado na Fig. 5.3.

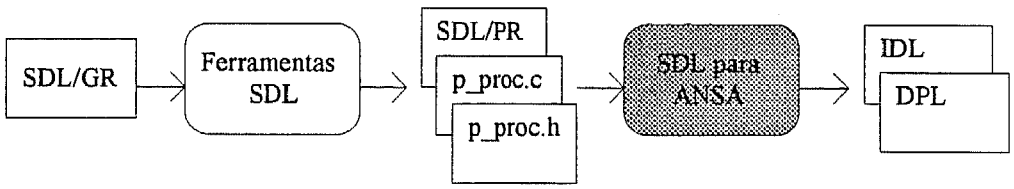


Fig. 5.3 - Processo de Tradução

O protótipo desenvolvido suporta as construções SDL necessárias à correspondência apresentada na Tabela 5.1. Este subconjunto da linguagem inclui:

Estrutura	Blocos, vias, tipos de processos e instâncias
Comunicação	Sinais
Procedimento	Estados, entrada e transições
Acção	Tarefas, criação, saídas e decisões.

Nesta primeira versão, os tipos de dados SDL suportados estão limitados aos tipos de dados pré-definidos e a construções como ARRAY, STRUCT e STRING que podem facilmente ser expressos em IDL e DPL.

5.2.2 Identificação de Objectos e Interfaces

A ideia geral de como identificar objectos e interfaces ANSAware numa especificação SDL foi descrita na secção 5.1. Seguidamente, é apresentada a tradução concreta para ANSAware.

Em ANSAware, operações e interfaces de um objecto são implementadas separadamente. Só as operações descritas em IDL são visíveis externamente, isto é, só estas podem ser acedidas através da interface. Para cada operação descrita na interface, uma operação correspondente deve estar presente no objecto que implementa as operações.

Cada processo SDL dá origem a um objecto ANSAware. Os resultados da tradução de um processo consistem em duas partes, uma que contém a interface do objecto ANSAware, e outra que contém a implementação das operações.

As interfaces são obtidas a partir das vias do processo tipo, como descrito anteriormente. Vias com a direcção de entrada em processos tipo correspondem a interfaces dos servidores e a lista de sinais existentes na definição das vias correspondem às operações que os servidores oferecem nessas interfaces. Para cada via de entrada é gerada, em IDL, uma descrição correspondente, e a partir desta a ferramenta ANSAware irá gerar rotinas C (*stubs*) que implementam a interface (secção 4.2.2).

Um exemplo para ilustrar a correspondência é o da Fig. 5.4.

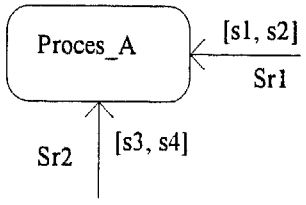


Fig. 5.4 - Processo SDL

O exemplo mostra um tipo de processo SDL com duas vias, cada uma com dois sinais que o processo aceita. A descrição IDL correspondente, apresentada na Fig. 5.5, define duas interfaces, uma para cada via.

```
Sr1 : INTERFACE =
BEGIN
        s1 : ANNOUNCEMENT OPERATION [lista_1] RETURNS [ ];
        s2 : ANNOUNCEMENT OPERATION [lista_2] RETURNS [ ];
END.

Sr2 : INTERFACE =
BEGIN
        s3 : ANNOUNCEMENT OPERATION [lista_3] RETURNS [ ];
        s4 : ANNOUNCEMENT OPERATION [lista_4] RETURNS [ ];
END.
```

Fig. 5.5 - Código IDL para as interfaces do Processo SDL

5.2.3 Processamento

Para suportar a implementação dos processos, é adicionada uma função ao ficheiro **.dpl** correspondente a cada processo. Como os processos SDL são descritos por máquinas de estados finitos (FSMs) (secção 3.2.4), a função C correspondente a cada processo é denominada FSMF (Função da Máquina de Estados Finitos). Esta função é obtida a partir dos ficheiros **.c** e **.h** do processo correspondente, gerados pela ferramenta SDL (secção 3.3.2).

A chamada da função FSMF é realizada nas funções que implementam as operações descritas na interface do objecto (secção 4.2.2) e no *body* do ficheiro **.dpl** desse objecto (secção 4.3).

Um exemplo da função FSMF é mostrado na Fig. 5.6. O primeiro *switch* testa o estado corrente, e o segundo testa a entrada recebida, podendo esta ser qualquer um dos sinais que o processo pode receber.

```
void Proces_A_FSMF (input)
{
 switch (current_state) {
    case state1:
        switch (input) {
            case s1:
                { /* código para a transição (state1,s1) */
                 NEXTSTATE (state2);
                }; break;
            case s2:
                { /* código para a transição (state1,s2) */
                 NEXTSTATE (state3);
                }; break;
        }
    case state2:
    . . .
 }
}
```

Fig. 5.6 - FSMF - Função da Máquina de Estados Finitos

5.2.4 Comunicação

Até aqui, os processos foram tratados de forma isolada. Vamos agora descrever como é que as interacções entre processos SDL são implementadas em ANSAware.

Em SDL existem basicamente duas formas de endereçamento, que podem ser usadas quando se envia um sinal: o **endereçamento explícito**, que exige que o PId do receptor do sinal seja conhecido à priori, sendo neste caso o sinal enviado a um dado processo identificado pelo PId; e o **endereçamento implícito** que é usado quando um sinal é enviado através de um canal, de uma via, ou a um processo não explicitamente identificado pelo PId.

Endereçamento Explícito

Em ANSAware, os PIds são implementados pelo conjunto de todas as referências de interfaces que existem para um dado objecto. Com base neste conjunto e no sinal em questão, a interface apropriada pode ser seleccionada. Implementar PIds como um conjunto de referências de interfaces significa que existe uma correspondência entre enviar PIds em SDL e passar conjuntos de referências de interfaces em ANSAware. A diferença é que referências de interfaces individuais não podem ser passadas.

Endereçamento Implícito

Em ANSAware é necessário que uma ligação seja estabelecida antes de qualquer invocação, isto é, que uma referência de interface seja importada antes de se fazer a chamada à operação desejada, o que não acontece em SDL neste tipo de endereçamento.

Quando traduzimos SDL para ANSAware, devemos garantir que as definições de canal e de via na especificação SDL são traduzidas em descrições correspondentes na implementação ANSAware. Isto deve-se à necessidade de determinar o receptor do sinal enviado sem um PId explícito. Uma vez que não serão estabelecidas ligações em ANSAware correspondentes às ligações SDL, as vias e os canais serão vistas como restrições às possíveis ligações em ANSAware, o que é implementado usando o *Trader* sempre que possível. Para além do tipo de interface, o *Trader* permite adicionar propriedades a uma dada oferta, isto é, quando se procura uma dada oferta um conjunto de restrições pode ser usado. Estas restrições devem ser verificadas pelas propriedades de qualquer oferta que seja retornada. Combinando este mecanismo propriedades/restrições com a hierarquia, pode-se restringir a verificação de ofertas no *Trader* àquelas que satisfazem as restrições impostas pelas vias e canais SDL.

Isto significa que serão usadas propriedades e restrições ANSAware para garantir que as ligações feitas em ANSAware são válidas no que diz respeito às ligações (canais e vias) SDL. Por exemplo, se duas instâncias de processos, cliente e servidor, não estiverem interligados em SDL, a implementação ANSAware do cliente não deve ser capaz de obter a referência das interfaces exportadas pelo servidor.

A solução proposta resulta no uso de vias e canais como pontos de referência. Dois processos podem comunicar um com o outro se estiverem ligados ao mesmo ponto de referência com a mesma direcção do sinal.

As propriedades e restrições de que uma instância de processo necessita quando exporta interfaces e importa referências de interfaces serão obtidas a partir das ligações SDL que ligam uma instância de um processo ao resto do sistema SDL.

Outro problema é como enviar um dado sinal a um de vários receptores possíveis, como é feito quando se envia um sinal através de um canal em SDL.

Enviar um sinal corresponde à invocação de uma operação de *announcement* em ANSAware [Tabela 5.1]. O processo que envia o sinal sem endereçamento explícito tem que importar, em ANSAware, uma interface de um objecto que suporta tal operação e com qual lhe é permitido comunicar.

Para solucionar este problema, o *Trader* foi novamente utilizado. Assim, o *Trader* é usado para seleccionar uma interface de entre as interfaces existentes que contêm a operação correspondente ao sinal, e estejam associadas ao objecto, que está de acordo com as restrições referidas anteriormente.

Em ANSAware não é possível importar do *Trader* uma referência de interface com base nas operações que esta pode fornecer. Esta restrição implica a necessidade de construir um tipo de hierarquia onde uma referência interface (correspondente a uma via de entrada num processo) provem das operações que oferece (sinais de entrada no processo). Esta hierarquia é conseguida declarando uma interface para cada sinal e sendo a interface, que implementa a via, compatível (Secção 4.2.2 - Compatibilidade de Interfaces) com as interfaces correspondentes aos sinais nela referidos. Desta forma, quando um cliente importa do *Trader* uma referência de interface correspondente à operação que deseja invocar, a referência que lhe é fornecida é a da interface do servidor que é compatível com a interface da operação desejada.

Considerando novamente o exemplo apresentado na Fig. 5.4 e se o endereçamento for implícito, o código IDL necessário é o apresentado na Fig. 5.7.

```
s1 : INTERFACE =
BEGIN
     s1 : ANNOUNCEMENT OPERATION [lista_1] RETURNS [ ];
END.


s2 : INTERFACE =
BEGIN
     s2 : ANNOUNCEMENT OPERATION [lista_2] RETURNS [ ];
END.


Sr1 : INTERFACE =
IS COMPATIBLE WITH s1;
IS COMPATIBLE WITH s2;
BEGIN
END.


s3 : INTERFACE =
BEGIN
     s3 : ANNOUNCEMENT OPERATION [lista_3] RETURNS [ ];
END.


s4 : INTERFACE =
BEGIN
     s4 : ANNOUNCEMENT OPERATION [lista_4] RETURNS [ ];
END.


Sr2 : INTERFACE =
IS COMPATIBLE WITH s3;
IS COMPATIBLE WITH s4;
BEGIN
END.
```

Fig. 5.7 - Código IDL quando o endereçamento é implícito

Para esclarecer melhor o conceito de restrições impostas pelo SDL e o uso de endereçamento implícito, consideremos o exemplo em que as ligações existentes na especificação SDL são as representadas na Fig. 5.8.

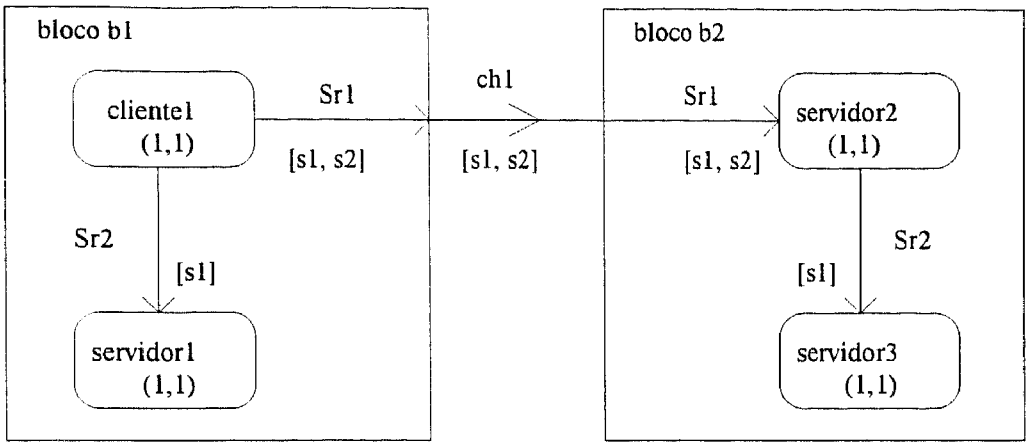


Fig. 5.8 - Interligações SDL

Neste exemplo, existem dois blocos interconectados e cada um deles contém dois processos. O processo **cliente1** envia o sinal **s1**, que pode ser recebido pelo **servidor1** e pelo **servidor2**, e o sinal **s2**, que só pode ser recebido pelo **servidor2**.

As interligações SDL resultam nas seguintes propriedades e restrições para os objectos:

servidor1

Esta instância de processo tem uma via de entrada, que está conectada a outra instância de processo do mesmo bloco. Quando o objecto correspondente a este processo exportar a sua interface para o *Trader*, deve usar a seguinte propriedade:

bloco b1 externo { } local {sr2}.

Esta propriedade significa que os valores de **bloco**, **externo** e **local** são, respectivamente, b1, um conjunto vazio e {sr2}.

servidor2

Esta instância de processo tem uma via de entrada, que está conectada a um canal. A seguinte propriedade deve ser usada quando o objecto correspondente ao processo exportar a sua interface:

bloco b2 externo {ch1} local { }.

Este servidor pode aceder ao servidor3 através da via Sr2. Para aceder a este servidor devem ser usadas as seguintes restrições quando importar a interface do *Trader*:

bloco=='b2' and 'sr2' in local.

Esta restrição significa que o valor de **bloco** deve ser b2 e o valor sr2 deve pertencer a **local**.

servidor3

Esta instância de processo é do mesmo tipo que o servidor1, mas está colocada noutro bloco. Quando o objecto correspondente a este processo exportar a sua interface para o *Trader*, deve usar a seguinte propriedade:

bloco b2 externo { } local {sr2}.

cliente1

Esta instância de processo possui duas vias de saída, uma conectada a um canal de saída e a outra a uma instância de um processo. As restrições a serem usadas quando este cliente importar uma interface são:

bloco=='b1' and 'sr2' in local or bloco!='b2' and 'ch1' in externo.

Esta restrição garante que a importação de uma referência de interface que aceite s1 não resultará na referência de interface do servidor3.

Trading

Dadas as propriedades do servidor1, servidor2 e servidor3 e as restrições do cliente1, a reacção do *Trader* aos possíveis pedidos de importações de referências de interfaces será:

* Quando o cliente1 importa do *Trader* uma referência de interface s1, é-lhe retornada uma referência de interface do servidor1 ou do servidor2.

* Quando o cliente1 importa do *Trader* uma referência de interface s2, é-lhe retornada a referência de interface do servidor2.

* Quando o servidor2 importa do *Trader* uma referência de interface s1, é-lhe retornada a referência de interface do servidor3.

Capítulo 6

Exemplo de Criação de um Serviço

Para uma melhor compreensão da correspondência apresentada anteriormente, neste capítulo é exemplificada a criação de um serviço simples de *News Multimedia* (NMM). Este serviço permite a visualização, a pedido de um cliente, de um documento multimédia, mais especificamente, de um documento HTML (*Hypertext Markup Language*).

A especificação consiste, em traços gerais, num cliente que irá solicitar a um servidor a localização de um documento. Após a obtenção, por parte do cliente, da localização do documento pretendido, a sua visualização é conseguida invocando uma aplicação, já existente, que após a descodificação do documento o apresenta no écran. O esboço do serviço é apresentado na Fig. 6.1.

A criação deste (ou de qualquer outro) serviço é realizada em três etapas (Secção 5.2.1):

- Especificar o serviço em SDL;
- Gerar os ficheiros necessários para a tradução;
- Traduzir esses ficheiros para ficheiros suportados pela plataforma ANSA;

Após a obtenção dos ficheiros ANSA (ficheiros **.idl** e **.dpl**) a aplicação está apta a ser linkada e compilada na plataforma ANSA, e finalmente executada.

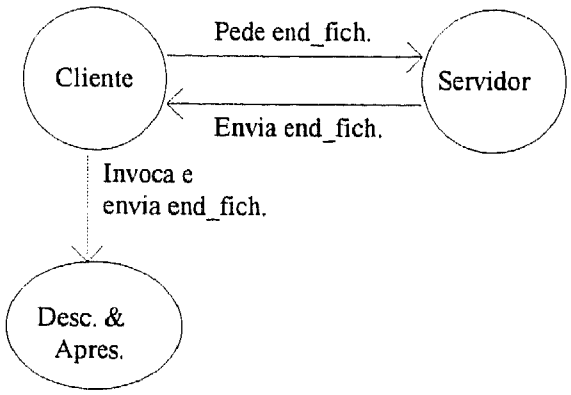


Fig. 6.1 - Serviço de *News Multimedia*

6.1 Especificação do Serviço

O nível mais alto de uma especificação SDL é o sistema, como foi visto na Secção 3.2. Neste exemplo o sistema, que tem como nome *NMM*, é constituído por um bloco (*bl*) que por sua vez contém dois processos (*cliente* e *servidor*) [Fig. 6.2].

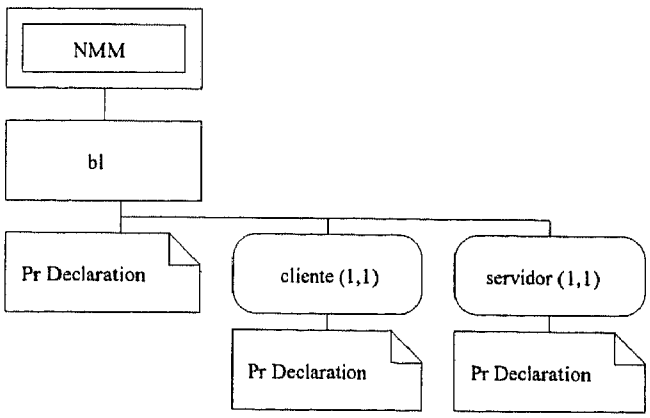


Fig. 6.2 - Diagrama hierárquico da especificação

No nível bloco os dois processos comunicam um com o outro através da via *Sr1* [Fig. 6.3]. O processo *cliente* envia o sinal *p* ao processo *servidor* e este envia ao *cliente* o sinal *r*.

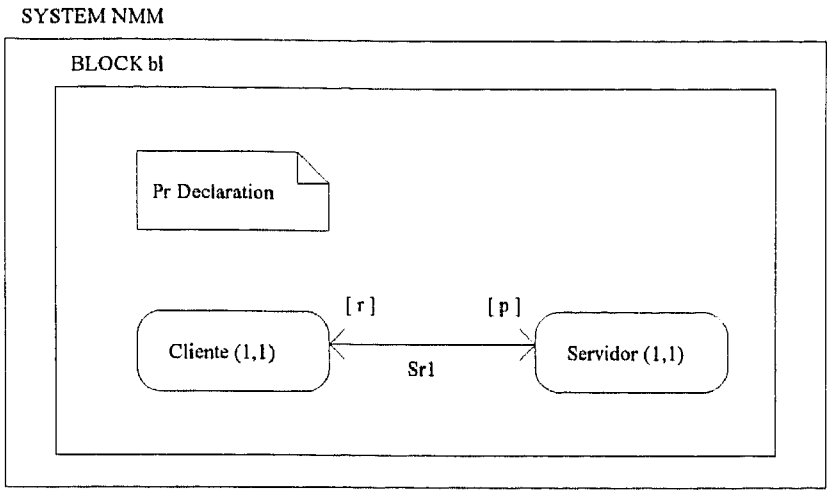


Fig. 6.3 - Diagrama de interligação

No nível processo, o processo *cliente* [Fig. 6.4] envia o sinal *p* contendo o nome do documento que pretende visualizar; depois de receber a localização do documento (sinal *r* que contém a localização) invoca um descodificador responsável pela apresentação indicando a localização do documento pretendido. Em termos de especificação SDL, a invocação à interface gráfica é realizada através do uso de um operador ADT (tipo de dado abstracto). A declaração deste ADT e de todas as variáveis utilizadas neste processo é mostrada na Fig 6.5.

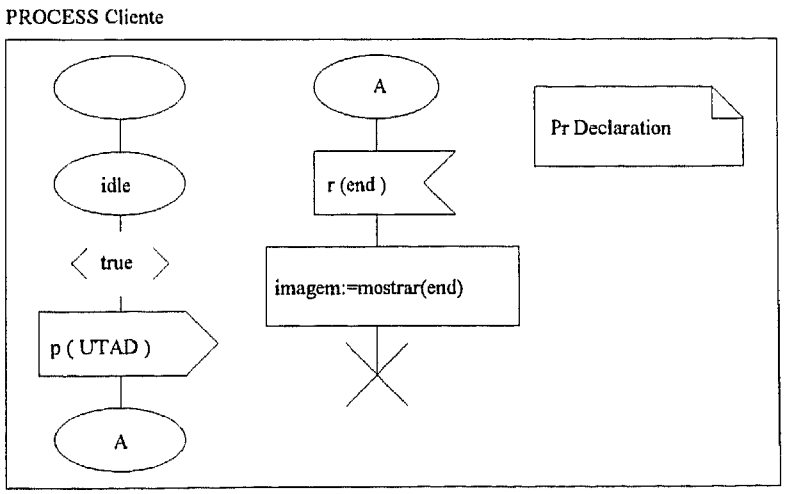


Fig. 6.4 - Processo cliente

```
Newtype img
Operators
    mostrar : Charstring -> img;
EndNewtype;

dcl end Charstring;
dcl imagem img;
```

Fig 6.5 - Declaração de variáveis do processo cliente

A especificação do processo *servidor* é apresentada na figura seguinte.

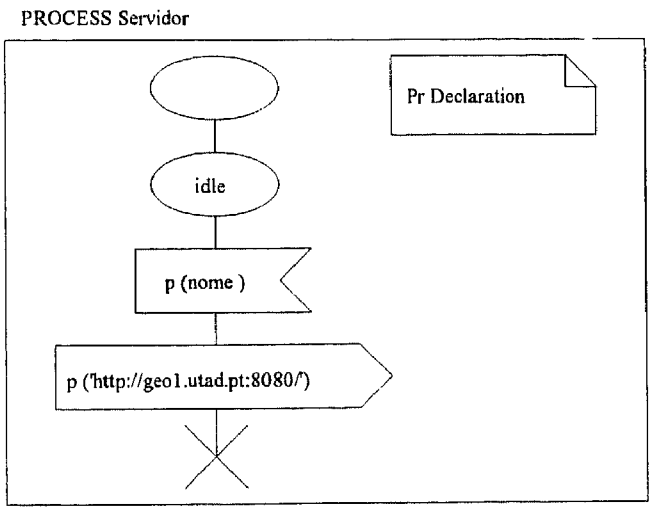


Fig. 6.6 - Processo servidor

6.2 Geração dos Ficheiros para a Tradução

Os ficheiros necessários para a tradução de SDL para ANSA, como referido anteriormente (Secção 5.2.1), são:

- o ficheiro da descrição na forma textual (**NMM.pr**);

- ficheiros **.h** e **.c** de cada processo (**p_cliente.h**, **p_cliente.c**, **p_servidor.h** e **p_servidor.c**).

O ficheiro de descrição **.pr** é gerado automaticamente pelo GEODE, aquando da especificação em modo gráfico. Este ficheiro não é mais do que a tradução da especificação em modo gráfico para modo texto.

Os ficheiros **.c** e **.h** dos processos *Cliente* e *Servidor* são obtidos utilizando o Gerador de Aplicações do GEODE. Como referido no Capítulo 3 (Secção 3.3.1), este Gerador necessita do ficheiro de descrição verificado (**NMM.gr**), construído a partir da forma textual (**NMM.pr**) usando a ferramenta **geodeverif**, e do ficheiro de configuração (**NMM.cfg**), construído pelo utilizador descrevendo a arquitectura física da aplicação.

O Gerador GEODE não gera código C para o operador ADT. Este código é fornecido pelo utilizador.

6.3 Tradução de SDL para ANSA

Como o ANSA requer descrições separadas para as interfaces e para a parte operacional dos objectos, a tradução consiste em duas partes:

- gerar as descrições das interfaces (ficheiros **.idl**);
- gerar a parte operacional dos objectos (ficheiros **.dpl**).

Os ficheiros **.idl** são gerados a partir do ficheiro **NMM.pr** e os **.dpl** a partir dos ficheiros **NMM.pr**, **p_cliente.c**, **p_cliente.h**, **p_servidor.c** e **p_servidor.h**.

A correspondência desta especificação SDL para ANSA é mostrada na figura seguinte:

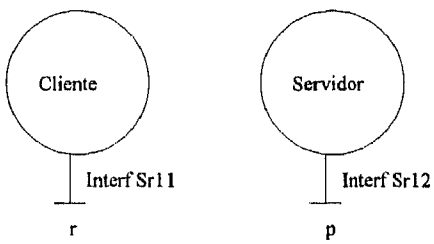


Fig. 6.7 - Correspondência SDL-ANSA

A tradução resulta em dois objectos ANSA, *cliente* e *servidor*, cada um com uma interface. A interface do objecto cliente corresponde à via de entrada no processo SDL correspondente e fornece a operação *r* (sinal de entrada no processo cliente). De igual modo o objecto servidor (correspondente ao processo servidor SDL) fornece a operação *p* na interface *Sr12*.

O resultado desta correspondência faz com que existam quatro ficheiros **.idl** (ver Secção 5.2.4) e dois **.dpl**. A estes últimos **.dpl**, para além das declarações PREPC (Secção 4.2.6) necessárias para declarar interfaces, importar e exportar do *Trader* interfaces, etc., é ainda adicionada uma função correspondente à máquina de estados finitos do respectivo processo. A chamada a esta função (FSMF) é realizada nas funções que implementam as operações descritas na interface do objecto e no *body* do ficheiro **.dpl** correspondente. Isto é, a chamada à função FSMF do cliente (p_cliente_FSMF) é realizada na função que implementa a operação *r* e no *body* do ficheiro **cliente.dpl**, e a chamada à função FSMF do servidor (p_servidor_FSMF) é realizada na função que implementa a operação *p* e no *body* do ficheiro **servidor.dpl**.

Nas figuras 6.8 e 6.9 são apresentadas, respectivamente, as funções correspondentes aos processos servidor e cliente.

```
void p_servidor_FSMF(int cur_sig)
{
switch((int)CURRENT_STATE) {

 case sta_start:
                switch(cur_sig) {
                  case sig_START:
                              CURRENT_STATE=sta_idle;
                  default: break;
                } break;

 case sta_idle:
                switch(cur_sig) {
                  case sig_p):
! (ifref2) <- traderRef$Import("r","/ansa/testservices","bloco=='bl'/
                              and 'srl' in local")
! () <- ifref2$p(" http://geol.utad.pt:8080:")
                  default: break;
                } break;
}
}
```

Fig. 6.8 - FSMF do processo servidor

```
void p_cliente_FSMF(int cur_sig)

ansa_String ver;

{
switch((int)CURRENT_STATE) {

 case sta_start:
                switch(cur_sig) {
                  case sig_START:
                              CURRENT_STATE=sta_idle;
                              if(CURRENT_STATE<=STA_LAST_CONTINUOUS)
                                  p_cliente_FSMF(sig_EMPTYQ);
                  default: break;
                } break;

 case sta_idle:
                switch(cur_sig) {
                  case sig_EMPTYQ):
! {ifref2} <- traderRef$Import("p","/ansa/testservices","bloco=='bl'/
                              and 'srl' in local")
! {} <- ifref2$p("UTAD")
                                  CURRENT_STATE=sta_a;
                  default: break;
                } break;

 case sta_a:
                switch(cur_sig) {
                  case sig_r:
                              sprintf(ver,"%s","xmosaic");
                              sprintf(ver+8,"%s",end_r);
                              system(ver);
                  default: break;
                } break;
 }
}
```

Fig. 6.9 - FSMF do processo cliente

Capítulo 7

Conclusões

Com o aparecimento de dispositivos de armazenamento de elevada capacidade, *workstations* com grande poder de processamento, técnicas de compressão, redes de grande velocidade e *software* apropriado, criar uma variedade de serviços de comunicação multimédia em ambientes distribuídos torna-se não só tecnicamente possível mas também economicamente viável. Por outro lado, com o aumento da competitividade dos mercados torna-se cada vez mais necessário criar serviços de uma forma rápida e a um custo relativamente baixo.

O trabalho desenvolvido teve por objectivo investigar a possibilidade de uma criação rápida de serviços em ambientes distribuídos. Para tal, foram estudados e apresentados, ao longo desta dissertação, vários aspectos decisivos que levaram à implementação do protótipo apresentado para essa criação.

Uma vez que multimédia é uma das formas mais inovadoras do uso de uma rede para aceder a informação e obter comunicações entre pessoas para além de que se espera que os sistemas multimédia corram em arquitecturas completamente distribuídas onde os dados multimédia são armazenados em diferentes locais, neste trabalho foram apresentadas as características e requisitos das comunicações multimédia. Foi feita ainda uma classificação dos serviços de comunicação dirigidos ao tratamento de informação multimédia, bem como uma apresentação de modelos, baseada no conceito de comunicação cliente/servidor, para o armazenamento e

transmissão de informação multimédia e para a localização de informação em sistemas distribuídos.

Como o SDL (*Specification Description Language*) é uma linguagem que permite descrever e especificar serviços, de uma forma fácil e rápida, e o ANSA (*Advanced Network System Architecture*) é uma arquitectura para ODP (*Open Distributed Processing*) que suporta o desenvolvimento e construção de aplicações distribuídas e portanto poderiam contribuir decisivamente para o objectivo, foi feita uma apresentação de dois pacotes de software: o GEODE (ambiente SDL) e o ANSAware (exemplo de implementação da arquitectura ANSA). Estes dois pacotes de software possuem ferramentas poderosas que quando combinadas são de grande utilidade para a criação rápida de serviços em ambientes distribuídos.

Para além da apresentação das suas ferramentas foram descritos os conceitos mais importantes, SDL e ANSA, bem como a forma como é gerado o código de uma aplicação especificada em SDL. Para a junção das ferramentas foram indicadas possíveis correspondências entre os seus conceitos. Com base nas vantagens e desvantagens que estas possíveis correspondências apresentavam, foi seleccionada uma que por fim levou à implementação do protótipo para o tradutor.

A correspondência usada tem como objectivo preservar tanto quanto possível a semântica original SDL, pois uma especificação SDL deve ser a representação do problema e não deve ser afectada por emulações de conceitos da infraestrutura destino. A descrição das relações entre os conceitos SDL e ANSAware, mostra que este objectivo foi atingido, pois as noções mais importantes da plataforma ANSAware, como objecto, interface, referência de interface e operações de *announcement*, têm uma correspondência clara a noções SDL.

Perspectivas Futuras

As ferramentas e os métodos de criação de serviços devem ajudar a encontrar consistência e qualidade apropriadas na próxima geração de sistemas e serviços em ambientes distribuídos.

Sistemas distribuídos é uma área que se encontra ainda em evolução, os primeiros passos foram dados, mas ainda existe um grande caminho a percorrer [Tanenbaum92]. Espera-se que os sistemas distribuídos se venham a tornar, de uma forma crescente, importantes nos próximos anos. De facto, provavelmente dentro de poucos anos, a maioria das organizações irá interligar a maior parte dos seus

computadores a grandes sistemas distribuídos de forma a fornecer melhores serviços, mais baratos e mais apropriados aos utilizadores.

Por outro lado, apesar da longa história do SDL, só recentemente teve início a era das Linguagens de Especificação Formal. Após o lançamento da actual versão (SDL-88), houve um período de estudo entre 1989 e 1992 realizado pelo extinto CCITT, cujo resultado foi uma linguagem completamente orientada a objectos (SDL - - 92) para ser fornecida como a nova recomendação Z.100 [Faergemand93].

A evolução na área dos sistemas distribuídos e consequentemente em plataformas distribuídas, como a apresentada neste trabalho (ANSAware), bem como a evolução de linguagens de especificação e descrição (SDL), irão implicar, de uma forma directa, a evolução natural do trabalho apresentado. Porque métodos e ferramentas de criação de serviços que ofereçam a possibilidade de uma criação rápida, com menores custos, e cujos serviços ofereçam um nível de qualidade apropriado e possam ser sujeitos a posteriores alterações irão cada vez mais ser utilizados, devido ao aumento quer da complexidade dos sistemas quer da competitividade dos mercados.

Anexo A

Tipos de Dados SDL

O SDL fornece um conjunto de tipos de dados pré-definidos e um conjunto de geradores pré-definidos que permitem que o utilizador especifique novos tipos.

Neste anexo são apresentados os tipos de dados pré-definidos bem como exemplos de especificações de novos tipos de dados [Belina91].

Tipos pré-definidos:

Boolean

Descrição: Variável lógica - **true** | **false**.

Declaração SDL: `Boolean`

Character

Descrição: Caracteres definidos pelo *International Alphabet no 5*.

Declaração SDL: `Character`

Charstring

Descrição: Conjunto de caracteres de qualquer tamanho.

Declaração SDL: `Charstring`

Integer

Descrição: Inteiro desde -∞ a +∞.

Declaração SDL: `Integer`

Natural

Descrição: Inteiro positivo.

Declaração SDL: `Natural`

Real

Descrição: Numeros reais desde -∞ a +∞.

Declaração SDL: `Real`

PId

Descrição: Endereço de uma instância de processo.

Declaração SDL: `PId`

Duration

Descrição: O mesmo que Real. O valor de uma variável deste tipo é a diferença entre dois valores do tipo Time.

Declaração SDL: `Duration`

Time

Descrição: O mesmo que Real. O valor de uma variável deste tipo representa um instante de tempo.

Declaração SDL: `Time`

Novos tipos:

Array

Um array pode ser especificado através de um gerador pré-definido (*Array*), da seguinte forma:

```
newtype Box
      Array (Character, Integer)
endnewtype Box;
```

Deste modo é definido um array do tipo *Box*, cuja indexação é feita através do tipo *Character*. Os seus valores são do tipo *Integer* e são acedidos, como em linguagens de programação, escrevendo *b(z)*, onde *b* é uma variável do tipo *Box*, *z* do tipo *Character* e *b(z)* do tipo *Integer*.

Structure

Podemos especificar uma estrutura em SDL como se mostra no exemplo seguinte:

```
newtype Pessoa
      struct
            nm    Charstring;
            nr    Integer;
            fm    Boolean;
endnewtype Pessoa;
```

Este novo tipo define uma estrutura *Pessoa* com os campos *nm* (nome), *nr* (número de identificação) e *fm* (feminino corresponde a true e masculino a false).

Set

Este tipo de dados pode ser especificado através de um gerador pré-definido (*Powerset*), da seguinte forma:

```
newtype Inteiroset
      Powerset (Integer)
      adding operators
            val : Inteiroset -> Integer;
            /*retorna o valor de Inteiroset*/
endnewtype Inteiroset;
```

Está definido o novo conjunto Inteiroset, cujos valores são do tipo Integer. Para este tipo todos os operadores usuais são válidos (operadores de Powerset), com a adição do novo operador val, que retorna um valor arbitrário de Inteiroset.

Enumeration

Uma enumeração pode ser especificada da seguinte forma:

```
newtype Item
      literals café, chocolate, bolachas;
      operators
            ordering;
            item_valor: Item -> Integer;
endnewtype Item;
```

Todos os operadores, exceptuando igualdade ou desigualdade, devem ser definidos explicitamente. Este exemplo mostra uma especificação (opcional) de ordenação através da palavra **ordering**. Isto faz com que os operadores <, >, <= e >= sejam válidos para os **literals** definidos. Ainda neste exemplo é feita uma especificação simples de um operador (item_valor). Este operador aceita um valor do tipo Item e retorna o seu valor correspondente do tipo Integer.

Anexo B

Ficheiro de Configuração SDL

O Gerador de Aplicação GEODE usa o ficheiro de configuração (*descrição*.**cfg**), que descreve a arquitectura física da aplicação, além da sua descrição SDL. Este ficheiro de configuração pode ser criado, modificado ou lido pelo utilizador em qualquer editor de texto. A sintaxe deste ficheiro é bem definida [VERILOG93d].

A sintaxe usada nos ficheiros de configuração bem como um exemplo deste tipo de ficheiro são apresentados neste anexo.

Convenções

bold	linguagem
itálico	referência a um identificador
<regra>::=	definição de uma regra
<regra>	referência a uma regra
[<regra>]	regra opcional
|	alternativa
<regra>*	regra cuja ocorrência está compreendida entre 0 e infinito
<regra>+	regra cuja ocorrência está compreendida entre 1 e infinito
{}	factorização

Sintaxe

```
<Conf>::=                          #BEGIN [<Num>] {<No>+
                                   {{<Tarefa> | <Grupo>}+ <Proc>*}*
<Externo>*                         }* #END
<Num>::=                           #RANGE Natural
<No>::=                            #NODE <Nome_Bloco_ou_Nome_Sistema>
<Tarefa>::=                        #TASK <Nome_Bloco_ou_Nome_Sistema>
                                   <Prioridade> <Tcod> [<Fich_Ext>]
<Grupo>::=                         #GROUP <Nome_Bloco_ou_Nome_Sistema>
                                   <Gcod>
<Proc>::=                          #PROC <Nome_Proc> <Prioridade>
                                   [<Fich_Ext>]
<Externo>::=                       #EXTERN <Nome_Tarefa_Ext>
                                   <Prioridade> -<Tcod> [<Fich_Ext>]
<Nome_Proc>::=                     Identifier I <Nome_Path>
<Nome_Bloco_ou_Nome_Sistema>::=    Identifier I <Nome_Path>
<Nome_Tarefa_Ext>::=               Identifier
<Fich_Ext>::=                      File Name Identifier
<Prioridade>::=                    Natural
<Tcod>::=                          Natural
<Gcod>::=                          Natural
<Nome_Path>::=                     {<Slash> Identifier}+
<Slash>::=                         /
```

Semântica

<Conf> descreve a aplicação completa para correr ou para simular.

<Num> define o número máximo de tarefas por nó. Este número deve ser maior que o número máximo de processos num nó.

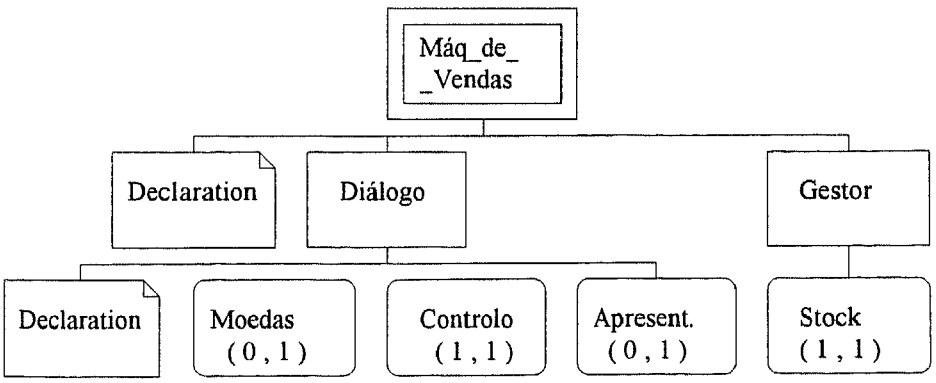
<No> o bloco (ou sistema) assim chamado designa o processador ou de uma forma mais geral o nó no sistema distribuído.

<Tarefa> indica que o bloco (ou sistema) assim chamado é a tarefa do executivo em tempo real destino (mapeamento TB).

<Grupo> indica que cada processo tipo no bloco (ou sistema) assim chamado é a tarefa do executivo em tempo real destino (mapeamento TP).
<Proc> descreve as características do tipo do processo SDL.
<Externo> descreve uma tarefa externa do executivo do nó previamente descrito. Usado quando a implementação do exterior do sistema SDL é uma tarefa externa.
<Nome_Tarefa_Ext> indica o nome lógico associado a uma tarefa externa.
<Fich_Ext> fornece o nome (ou a *path*) do ficheiro de uma tarefa escrita pelo utilizador
<Tcod> indica o número de código associado a uma tarefa externa; este código é usado para a determinação de chaves para várias fontes associadas com essa tarefa.
<Gcod> indica o número de código associado ao Grupo GEODE; este código é usado para determinar o número do nó numa aplicação com múltiplos nós (mapeamento TP).

EXEMPLO:

Consideremos um exemplo de uma descrição SDL, cujo diagrama hierárquico é o mostrado na figura seguinte:



O ficheiro de configuração correspondente a este exemplo pode ser:

```
#BEGIN
#NODE Máq_de_Vendas
      #GROUP Máq_de_Vendas 1
#END
```

Anexo C

Comando de Geração da Aplicação

Durante a etapa de geração da aplicação, ficheiros fonte, de *include* e *makefile* são gerados na plataforma GEODE.

O Gerador de Aplicação GEODE é invocado usando o seguinte comando [VERILOG93d]:

geodeb_rte *rte descrição*
[-same]
|
[[-user | -inter [*ficheiro* **[.adt]]] [-t] [-chk]**
[-prefix *xxx* **] [-fprefix** *ficheiro* **[.prf]]**
]

Onde:

rte

Representa o Executivo em Tempo Real no qual a aplicação irá ser executada. Este parâmetro pode ter um dos seguintes

valores (a lista não é exaustiva): **unix, psos, mtos, ose, vrtx, rtc.**

descrição

Representa os ficheiros de entrada contendo a descrição do sistema verificada e o ficheiro de configuração.

Opções

-same

Usa as mesmas opções da geração feita anteriormente.

-user

Permite ao utilizador especificar num ficheiro (*ficheiro*.**adt**) a lista de ADTs cujos operadores serão implementados pelo utilizador. Os restantes operadores serão interactivos. Se nenhum ficheiro (**.adt**) é especificado com esta opção, todos os operadores serão interactivos.

-inter

Permite ao utilizador especificar num ficheiro (*ficheiro*.**adt**) a lista de ADTs cujos operadores serão interactivos. Todos os restantes serão implementados pelo utilizador. Se nenhum ficheiro **.adt** é especificado com esta opção, todos os operadores serão implementados pelo utilizador.

Se nenhuma destas opções (**-user** | **-inter**) é especificada, por omissão todos os operadores são implementados pelo utilizador.

-t

Requer geração de código com informações das instruções SDL.

-chk

Requer verificação nos dados (*array overflow*, divisão por zero, etc).

-prefix

Permite ao utilizador introduzir equivalências entre declarações SDL e declarações C.

-fprefix

Igual ao **-prefix**, com o ficheiro **.prf** contendo a lista de prefixos.

O número máximo de prefixos no ficheiro é de 50.

Anexo D

Tipos de Dados IDL

A Linguagem de Definição de Interfaces (IDL) suporta vários tipos de dados e construtores de tipos de dados, sendo estes últimos usados para construir estruturas de dados mais complexas a partir dos tipos de dados básicos.

Neste anexo são apresentados os tipos de dados IDL bem como os construtores [ANSA93b].

Boolean

Descrição: variavel lógica - **ansa_TRUE | ansa_FALSE**.

Declaração IDL:
```
BOOLEAN;
```
ou
```
MyBool : TYPE = BOOLEAN;
```

Declaração C:
```
typedef unsigned long int ansa_Boolean;
```

Short Cardinal

Descrição: inteiro não negativo com 16 bits de precisão.

Declaração IDL:
```
SHORT CARDINAL;
```
ou

```
MySCard : TYPE = SHORT CARDINAL;
```

Declaração C:
```
typedef unsigned short int ansa_ShortCardinal;
```

Cardinal

Descrição: inteiro não negativo com 32 bits de precisão.

Declaração IDL:
```
CARDINAL;
```
ou
```
MyCard : TYPE = CARDINAL;
```

Declaração C:
```
typedef unsigned long int ansa_Cardinal;
```

Long Cardinal

Descrição: inteiro não negativo com precisão > 32 bits.

Declaração IDL:
```
LONG CARDINAL;
```
ou
```
MyLCard : TYPE = LONG CARDINAL;
```

Declaração C:
```
typedef unsigned long int ansa_LongCardinal;
```

Short Integer

Descrição: inteiro positivo ou negativo com 16 bits de precisão.

Declaração IDL:
```
SHORT INTEGER;
```
ou
```
MySInt : TYPE = SHORT INTEGER;
```

Declaração C:
```
typedef short int ansa_ShortInteger;
```

Integer

Descrição: inteiro positivo ou negativo com 32 bits de precisão.

Declaração IDL:
```
INTEGER;
```
ou
```
MyInt : TYPE = INTEGER;
```

Declaração C:
```
typedef long int ansa_Integer;
```

Long Integer

Descrição: inteiro positivo ou negativo com precisão > 32 bits.

Declaração IDL:
```
LONG INTEGER;
```
ou
```
MyLInt : TYPE = LONG INTEGER;
```

Declaração C:
```
typedef long int ansa_LongInteger;
```

Real

Descrição: real expresso com 32 bits.

Declaração IDL:
```
REAL;
```
ou
```
MyReal : TYPE = REAL;
```

Declaração C:
```
typedef float ansa_Real;
```

Long Real

Descrição: real expresso com 64 bits.

Declaração IDL:
```
LONG REAL;
```
ou
```
MyLReal : TYPE = LONG REAL;
```

Declaração C:
```
typedef double ansa_LongReal;
```

Octet

Descrição: quantidade com 8 bits.

Declaração IDL:
```
OCTET;
```
ou
```
MyOctet : TYPE = OCTET;
```

Declaração C:
```
typedef unsigned char ansa_Octet;
```

Char

Descrição: caracter ASCII de 7 bits.

Declaração IDL:
```
CHAR;
```
ou
```
MyChar : TYPE = CHAR;
```

Declaração C:
```
typedef char ansa_Char;
```

String

Descrição: sequência de caracters de tamanho variável.

Declaração IDL:
```
STRING;
```
ou
```
MyString : TYPE = STRING;
```

Declaração C:
```
typedef char *ansa_String;
```

Enumeration

Descrição: sequência de valores mapeada num intervalo de inteiros.

Declaração IDL:
```
Success : TYPE = {F,S};
```

Declaração C:
```
typedef enum {F,S} Success;
```

Array

Descrição: sequência de tamanho fixo de elementos de um dado tipo base.

Declaração IDL:
```
Address : TYPE = ARRAY 16 OF OCTET;
```

Declaração C:
```
typedef ansa_Octet Address[16];
```

Sequence

Descrição: sequência de tamanho variável de elementos de um dado tipo.

Declaração IDL:
```
Vector : TYPE = SEQUENCE OF REAL;
```

Declaração C:
```
typedef struct {
        ansa_ShortCardinal      length;
        ansa_Real   *data;
} Vector;
```

Record

Descrição: grupo de elementos de diferentes tipos base.

Declaração IDL:

```
HostEntry : TYPE = RECORD [
        H_Name : STRING,
        H_Type : INTEGER,
        H_Addr : Address
];
```

Declaração C:

```
typedef struct {
        ansa_String H_Name;
        ansa_Integer        H_Type;
        Address      H_Address;
} HostEntry;
```

Choice

Descrição: união descriminada.

Declaração IDL:

```
Result : TYPE = CHOICE Success OF {
        F       =>    INTEGER,
        S       =>    STRING
};
```

Declaração C:

```
typedef struct {
        Success designator;
        union {
              ansa_Integer       u_F;
              ansa_String u_S;
              } u;
} Result;
```

Alias

Descrição: um nome alternativo para um tipo.

Declaração IDL:

```
MyAddr : TYPE = Address;
```

Declaração C:

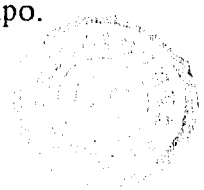
```
typedef Address MyAddr;
```

ansa_InterfaceRef

Descrição: referência a uma instância de uma interface de qualquer tipo.

Declaração IDL:

```
ansa_InterfaceRef;
```



Declaração C:

```
typedef struct {
        ansa_Cardinal       length;
        ansa_InterfaceRecord    *data;
} ansa_InterfaceRef;
```

InterfaceRef

Descrição: referência a uma instância de uma interface de um tipo particular.

Declaração IDL:

```
MyRef : INTERFACEREF OFTYPE X
```

Declaração C:

```
typedef ansa_InterfaceRef MyRef;
```

A informação **OFTYPE** é usada pelo PREPC para verificar se a referência da interface é do tipo correcto.

Bibliografia

[ANSA93a] Architecture Projects Management, *An Overview of ANSAware 4.1*, Doc. RM.099.02, February 1993.

[ANSA93b] Architecture Projects Management, *Application Programming in ANSAware 4.1*, Doc. RM.102.02, February 1993.

[Bartee89] Bartee, Thomas C., *ISDN, DECnet™, and SNA Communications*, HOWARD W. SAMS & COMPANY, 1989.

[Belina91] Belina, Ferenc and others, *SDL with Applications from Protocol Specification*, Prentice Hall International (UK) Ltd, 1991.

[Cole93] Cole, B., Interactive Multimedia: The Technology Framework. In: *IEEE Spectrum*, Vol. 30, No. 3, March 1993.

[Encontre89] Encontre, Vincent, GEODE: An industrial environment for Designing Real Time Distributed Systems in SDL. In: *SDL'89 The Language at Work*, Elsevier Science Publishers B. V., pp. 105 - 115, 1989.

[Faergemand93] Faergemand, Ove and Olsen, Anders, *New Features in SDL - - 92*, Telecommunications Research Laboratory, Denmark, February 1993.

[Habib92] Habib, Ibrahim W. and Saadawi, Tarek N., Multimedia Traffic Characteristics in Broadband Networks. In: *IEEE Communications Magazine*, Vol. 30, No.7, pp. 48 - 54, July 1992.

[Heijenk94] Heijenk, Geert J. and others, Communication Systems Supporting Multimedia Multi-user Applications. In: *IEEE NETWORK - The Magazine of Computer Communications*, Vol. 8, No 1, pp. 34 - 44, January/February 1994.

[Herbert94] Herbert, Andrew, An ANSA Overview. In: *IEEE NETWORK - The Magazine of Computer Communications*, Vol. 8, No 1, pp. 18 - 23, January/February 1994.

[Kretz92] Kretz, Francis and Colaitis, Françoise, Standardizing Hypermedia Information Objects. In: *IEEE Communications Magazine*, Vol. 30, No.5, pp. 60 - 70, May 1992.

[Loeb92] Loeb, Shoshana, Delivering Interactive Multimedia Documents over Networks. In: *IEEE Communications Magazine*, Vol. 30, No.5, pp. 52 - 59, May 1992.

[Mierop93] Mierop, John and others, Service Interaction in an Object-Oriented Environment. In: *IEEE Communications Magazine*, Vol. 31, No.8, pp. 46 - 51, August 1993.

[Mouftah92] Mouftah, H. T., Multimedia Communications: An Overview. In: *IEEE Communications Magazine*, Vol. 30, No.5, pp. 18 - 19, May 1992.

[Nunes92] Nunes, Mário Serafim e Casaca, Augusto Júlio, *Redes Digitais com Integração de Serviços*, Lisboa: Editorial Presença, 1992.

[Oliveira95] Oliveira, José Manuel Soares, *Concepção de Sistemas Quiosque Multimédia*, Dissertação de Mestrado, Faculdade de Engenharias da Universidade do Porto, Fevereiro 1995.

[Rakow95] Rakoww, T. and others, *Multimedia Database Systems - The Notions and the Issues*, Integrated Publication and Information Systems Institute, 1995.

[Rangan92] Rangan, P. Venkat and others, Designing an On-Demand Multimedia Service. In: *Communications Magazine*, Vol. 30, No.7, pp. 56 - 64, July 1992.

[Rosenberg92] Rosenberg, Jonathan and others, Multimedia Communications For Users. In: *IEEE Communications Magazine*, Vol. 30, No.5, pp. 20 - 37, May 1992.

[Rubin94] Rubin, Harvey and Natarajan, N., A Distributed Software Architecture for Telecommunication Networks. In: *IEEE NETWORK - The Magazine of Computer Communications*, Vol. 8, No 1, pp. 8 - 17, January/February 1994.

[Rumbaugh91] Rumbaugh, J. and others, *Object Oriented Modeling and Design*, Prentice - Hall, 1991.

[Shetler90] Shetler, T., The Birth of the BLOB. In: *Byte*, February 1990.

[Tanenbaum92] Tanenbaum, Andew S., *Modern Operating Systems*, Prentice - Hall, 1992.

[Timms89a] Timms, S., Broadband Communications: The Commercial Impact. In: *IEEE Network Magazine*, pp 10 - 15, July 1989.

[Timms89b] Timms, S., Broadband Communications: The Commercial Impact. In: *IEEE Infocom*, pp 602 - 210, 1989.

[VERILOG93a] VERILOG, *GEODE An Introduction to SDL and MSC*, Doc. D/GEXX/LA/100/340, October 1993.

[VERILOG93b] VERILOG, *GEODE EDITOR - Reference Manual*, Vers. 2.2, Doc. D/GEXX/RA/221/340, October 1993

[VERILOG93c] VERILOG, *GEODE C Application Generators; Reference Manual*, Vers. 2.1, Doc D/GCXX/RA/211/344, November 1993.

[VERILOG93d] VERILOG, *GEODE Simulator - Reference Manual*, Vers. 2.1, Doc. D/GSMX/RA/211/340, October 1993.

[Wright93] Wright, David, *Broadband: Business Services, Technologies, and Startegic Impact*, Artech House, 1993.

[Walters91] Walters, S. M., A New Direction for Broadband ISDN. In: *IEEE Communications Magazine*, Vol. 29, No.9, pp. 39 - 42, 1991.