Rogério Filipe Pinho Ferreira

# Metodologia de partilha de dados entre sistemas heterogéneos usando XML




Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2002

Rogério Filipe Pinho Ferreira

# Metodologia de partilha de dados entre sistemas heterogéneos usando XML




*Tese submetida à Faculdade de Ciências da Universidade do Porto para obtenção do grau de Mestre em Informática*

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2002

**Aos meus pais e ao meu irmão...**

# Agradecimentos

Em primeiro lugar gostaria de agradecer aos meus orientadores, José Paulo Leal e Fernando da Silva, pelo apoio que me deram durante o desenvolvimento do projecto. Em particular a gostaria de agradecer as instalações que me disponibilizaram no departamento para que tivesse as condições necessárias de trabalho.

Agradeço ao meu amigo Licínio Oliveira, meu colega de gabinete durante este projecto, com quem partilhei o dia a dia e me apoiou nos momentos mais difíceis.

For fim gostaria de agradecer de forma especial ao meus pais que permitiram que eu continuasse a estudar dando-me todo o apoio e condições necessárias para o desenvolvimento do meu trabalho sem nunca condicionarem a minha dedicação a tempo inteiro a este projecto.

# Conteúdo

**Índice de Tabelas** **11**

**Índice de Figuras** **14**

**Índice de Programas** **17**

**Resumo** **19**

**Abstract** **21**

**1 Introdução** **23**

1.1 Sistemas de informação . . . . . 24

1.2 Partilha de dados entre sistemas heterogéneos . . . . . 24

1.3 Metodologia para partilha de dados . . . . . 25

1.3.1 Objectivos . . . . . 25

1.3.2 Abordagem . . . . . 26

1.3.3 Exemplo . . . . . 28

1.4 Trabalho relacionado . . . . . 29

1.4.1 Conversão de dados . . . . . 30

1.4.2 Serviços de dados . . . . . 31

**2 Estado da Arte** **33**

2.1 UML . . . . . 33

2.1.1 Modelação . . . . . 35

2.1.2 Modelo conceptual . . . . . 40

2.2 XML . . . . . 46

2.2.1 Linguagens de anotação . . . . . 47
2.2.2 Núcleo do XML . . . . . 48
2.2.3 Convenções . . . . . 52
2.2.4 Meta-Linguagens . . . . . 54
2.2.5 Utilizações do XML . . . . . 61
2.2.6 Aplicações . . . . . 65
2.2.7 Processamento de documentos . . . . . 68
2.3 Serviços Web . . . . . 70
2.3.1 Comunicação entre aplicações . . . . . 73
2.3.2 Descrição de serviços . . . . . 76
2.3.3 Comunicação de dados . . . . . 76
2.3.4 Segurança . . . . . 81
2.3.5 Ambientes de desenvolvimento . . . . . 86

**3 Metodologia de partilha de dados** **89**

3.1 Modelação de dados . . . . . 89
3.1.1 Utilização do UML . . . . . 89
3.1.2 Exemplo . . . . . 94
3.2 Estruturas de Dados . . . . . 96
3.2.1 Definição . . . . . 97
3.2.2 Mapeamento . . . . . 100
3.2.3 Criação . . . . . 102
3.2.4 Exemplo . . . . . 105
3.3 Serviços de dados . . . . . 108
3.3.1 Definição do serviço . . . . . 109
3.3.2 A meta-informação . . . . . 113
3.3.3 Exemplo . . . . . 115
3.3.3.1 Servidor . . . . . 116
3.3.3.2 Implementação de segurança com o Axis . . . . . 125

**4 Conclusão** **127**

4.1 Análise da metodologia . . . . . 127
4.2 Análise da implementação . . . . . 128
4.3 Perspectivas futuras . . . . . 130
4.4 Opinião pessoal . . . . . 130

**A Anexos** **131**

A.1 WSDL . . . . . 131
A.2 Aplicações . . . . . 133
A.2.1 Servidor . . . . . 133
A.2.2 Cliente . . . . . 137

**Referências** **143**

# Índice de Tabelas


2.1 Tipos de diagrama UML. . . . . 43

2.2 Principais características do XML . . . . . 48

2.3 Principais Características do *XML Schema* . . . . . 62

2.4 Componentes principais dos *Schemas* . . . . . 63

2.5 Contributo do XML nos desafios da partilha de informação. . . . . 64

2.6 Requisitos de segurança e as respectivas soluções. . . . . 82

3.1 Atributos de uma classe que define um tipo de dados . . . . . 92

3.2 Regras de Mapeamento . . . . . 99

3.3 Informação do cabeçalho das mensagens SOAP. . . . . 114

3.4 Elementos que identificam os objectos pretendidos. . . . . 115

# Índice de Figuras


1.1 Troca de informação entre sistemas heterogéneos. . . . . 25

1.2 Conversores numa organização com 6 organismos. . . . . 26

1.3 Nova troca de informação entre dois sistemas heterogéneos. . . . . 27

1.4 Abordagem da metodologia. . . . . 28

1.5 Possível organização do Ministério da Educação. . . . . 29

2.1 Semelhanças entre um diagrama de classes UML e um grafo. . . . . 41

2.2 Classe UML. . . . . 43

2.3 Interface UML. . . . . 43

2.4 Colaboração UML. . . . . 44

2.5 Associação UML. . . . . 44

2.6 Dependência UML. . . . . 44

2.7 Generalização UML. . . . . 44

2.8 Realização UML. . . . . 45

2.9 Evolução das linguagens de anotação. . . . . 47

2.10 Árvore de nós do documento 2.1. . . . . 49

2.11 Visão do documento 2.10. . . . . 57

2.12 Posição do XMI relativamente a algumas aplicações. . . . . 67

2.13 Tecnologias que rodeiam os serviços Web. . . . . 72

2.14 Funcionamento de um pedido CORBA. . . . . 75

2.15 Estrutura de uma documento WSDL. . . . . 77

2.16 Estrutura da mensagem SOAP. . . . . 78

2.17 Exemplo de cabeçalho HTTP. . . . . 79

3.1 Exemplo de definição de um novo tipo de dados . . . . . 92
3.2 Definição do organismo Departamento de Pessoal. . . . . 94
3.3 Definição do organismo Secretaria. . . . . 95
3.4 Definição do organismo Departamento A. . . . . 96
3.5 Definição do sistemas global. . . . . 97
3.6 Processo de Mapeamento. . . . . 98
3.7 Comunicação de um cliente com um serviço Web. . . . . 110
3.8 Relação das secções de um documento WSDL. . . . . 110
3.9 Necessidade do documento WSDL por parte do cliente e do servidor. . . . . 116
3.10 Complexidade do uso dos PKI's iniciais. . . . . 124
3.11 Uso dos PKI's com serviços de confiança. . . . . 124

# Índice de Programas


2.1 Exemplo de um documento XML. . . . . 49

2.2 Documento XML. . . . . 50

2.3 DTD do documento XML 2.2 . . . . . 50

2.4 *Schema* do documento XML 2.2. . . . . 51

2.5 Documento com anotações de diferentes vocabulários. . . . . 52

2.6 Utilização de *namespaces*. . . . . 53

2.7 Documento XML usando SOAP e XHTML com *namespaces*. . . . . 53

2.8 Uso de uma expressão XPath. . . . . 54

2.9 Transformação XSLT para HTML. . . . . 55

2.10 Documento HTML resultante da transformação XSLT. . . . . 56

2.11 Variáveis no XSLT. . . . . 57

2.12 Elementos de iteração do XSLT. . . . . 57

2.13 Elemento para fazer testes condicionais no XSLT. . . . . 57

2.14 Importação de definições em XSLT. . . . . 57

2.15 Procedimento em XSLT. . . . . 57

2.16 Extensões em XSLT. . . . . 58

2.17 Geração de vários documentos usando XSLT. . . . . 58

2.18 Geração de XML em XSLT. . . . . 58

2.19 Utilização do XQuery. . . . . 60

2.20 Outra hipótese de utilização do XQuery. . . . . 60

2.21 Documento SOAP no corpo de uma mensagem HTTP. . . . . 79

2.22 Pacote HTTP na transmissão de uma mensagem SOAP. . . . . 80

2.23 Passagem de credenciais de segurança no URL. . . . . 84

2.24 Aplicação de um cliente SOAP em Java. . . . . 87
2.25 Aplicação de um cliente SOAP em Perl. . . . . 88
3.1 Excerto de um *Schema*. . . . . 99
3.2 Instância do *Schema* 3.1. . . . . 100
3.3 Definição de tipos identificadores. . . . . 101
3.4 Definição de tipos de dados definidos no modelo de dados. . . . . 102
3.5 Definição de uma tabela em SQL. . . . . 103
3.6 Definição de um interface em ODL. . . . . 104
3.7 Schema do organismo Departamento A. . . . . 105
3.8 Estrutura de dados em SQL. . . . . 107
3.9 Estrutura de dados em ODL. . . . . 108
3.10 Definições do WSDL. . . . . 110
3.11 Importações de definições do WSDL. . . . . 111
3.12 Definição das mensagens do WSDL. . . . . 111
3.13 Definição do tipo de porta do WSDL. . . . . 112
3.14 Definição do *binding* do WSDL. . . . . 113
3.15 Definição do serviço em WSDL. . . . . 113
3.16 *Schema* que define meta-informação das mensagens usado pelo documento WSDL. . . 114
3.18 Utilização da aplicação cliente. . . . . 119
3.19 Utilização da aplicação para fazer um pedido de informação. . . . . 119
3.20 Documento de resposta ao pedido. . . . . 119
3.21 Resposta ao pedido em HTML. . . . . 121
3.22 Instruções que definem o URL e nome do serviço. . . . . 122
3.23 Comando para disponibilizar o serviço. . . . . 125
3.24 Documento que refere o *handler* do lado do servidor. . . . . 125
3.25 Comando para disponibilizar o *handler* do lado do cliente. . . . . 126
3.26 Documento que faz a referência à chave. . . . . 126
A.1 WSDL do serviço implementado. . . . . 131
A.2 Aplicação Java do serviço. . . . . 133
A.3 Meta-informação no cabeçalho das mensagens SOAP. . . . . 134

A.4 Metodo que constrói o documento de resposta. . . . . 134
A.5 Aplicação do filtro sobre o documento de objectos. . . . . 136
A.6 Cria o elemento de raiz do documento de resposta. . . . . 136
A.7 Meta-informação no cabeçalho da resposta. . . . . 137
A.8 Método que trata dos parâmetros passados na linha de comandos. . . . . 137
A.9 Define o serviço por defeito. . . . . 139
A.10 Método que verifica o documento de resposta. . . . . 139
A.11 Método que grava o resultado em ficheiros. . . . . 140

# Resumo

Hoje em dia, os sistemas de informação desempenham um papel fundamental no apoio à gestão e funcionamento das organizações. Muitas das organizações são compostas por organismos parcialmente autónomos, cada um com a sua base de dados, pelo que o problema da partilha de dados entre os vários subsistemas é fundamental para se obter uma visão global e integrada do sistema de informação da organização. O problema da partilha de dados entre sistemas de dados heterogéneos, continua a ser um problema complexo de resolver.

Este trabalho propõe uma metodologia que determine uma solução, simples e eficaz, para o problema da partilha de dados. A metodologia envolve três etapas que determinam um percurso lógico para se chegar a uma solução. A primeira etapa consiste na definição do modelo abstracto de dados do sistema de informação global da organização, integrando de forma consistente os modelos do subsistemas. Neste modelo é ainda possível definir responsabilidade e direitos ao nível da geração e obtenção de informação. A segunda etapa envolve a geração automática de um formato de dados único a partir do modelo abstracto de dados. Este formato deve permitir a definição e validação dos dados. A terceira etapa envolve a definição de um modelo de comunicação para troca de dados entre os organismos, tirando partido do formato de dados definido.

A metodologia proposta baseia-se em *standards* para uniformizar as etapas de modelação, definição e comunicação de dados. A modelação de dados usa diagramas de classes UML, já que apenas pretendemos definir aspectos estáticos do sistema de informação da organização. Recorre-se a tecnologias XML, nomeadamente os *XML Schemas*, para definir o formato de dados. Esta definição de dados obtem-se por aplicação de uma folha de estilos XSLT sobre o modelo de dados em formato XMI. A comunicação de dados usa os serviços Web, permitindo tirar partido das tecnologias XML usadas na definição dos dados. A implementação dos serviços Web tem por base o protocolo de comunicação SOAP no qual se define o formato das mensagens a comunicar. O modelo de comunicação considera ainda questões de segurança na troca de dados.

Para demonstrar a aplicabilidade e eficácia da metodologia proposta, desenvolveu-se um cenário baseado num possível sistema de informação para uma Universidade e criaram-se dois protótipos, um cliente e um servidor, para demonstrar o funcionamento dos serviços Web. A escolha das ferramentas a usar no desenvolvimento e concretização da metodologia recaiu, propositadamente, sobre soluções abertas, gratuitas e multi-plataforma.

# Abstract

Nowadays, information systems with web access play a major role in the management and functioning of many organizations. Many of these organizations are internally structured in smaller, partially autonomous, subsidiary companies, each with possibly its own database, and therefore data sharing within the organization is fundamental in order to provide an integrated view of its information system. However, data sharing among heterogeneous data sources is still a complex problem.

This work proposes a methodology that determines a simple and workable solution for the data sharing problem. It involves three phases that define a logical path to get to a solution. The first phase is the definition of an abstract data model for the all information system, hence integrating in a consistent way the partial subsidiary systems. In this model one may also define responsibilities and rights within the information system. The second phase involves the automatic creation of a unified data format directly from the abstract model. Preferably, the format should allow for integrity checks to be defined within. The third phase, involves the definition of a data communication model that takes advantage of the data format defined previously and that allows for efficient data exchange among individual components of the system.

The methodology proposed is based on standards in order to standardize the data modeling, data definition and data communication phases. Since we are mainly interested in static aspects of the information system, UML diagrams are used to define the abstract data model. XML technologies, specially XML Schemas, are used to define the format for the data. This definition can be obtained by applying an XSLT style sheet over the XMI description of the data model. Web services are used for data communication, in particular we use the protocol SOAP for the messages being transmitted. We also discuss security regarding data communication.

To demonstrate the applicability and efficacy of the proposed methodology, we have developed a case study based on a possible information system for a University, and implemented two prototypes, one for the client and another for the server, to show the web services functioning. The tools adopted for the design and implementation were mainly those that were open source, free-ware and multi-platform.

# Capítulo 1

# Introdução

Na última década a Internet consolidou-se como a forma de comunicação de informação digital. Um utilizador dispondo dum vulgar computador pessoal com ligação à Internet e com clientes apropriados pode aceder a conteúdos multimédia, transferir ficheiros ou trocar mensagens com servidores em qualquer ponto do planeta.

Embora a comunicação entre utilizadores e servidores seja hoje uma questão fácil, a comunicação automática de dados é ainda um problema complexo. Este problema agrava-se se a comunicação de dados for entre sistemas heterogéneos, assentes em sistemas de gestão de bases de dados diferentes ou simplesmente se as estruturas de informação dos sistemas for diferente.

Numa organização com organismos autónomos o problema da comunicação de dados entre sistema heterogéneos torna-se particularmente importante. Idealmente as organizações disporiam dum único sistema de informação, com ramificações nos vários organismos. Na realidade é frequente que cada organismo tenha desenvolvido um sistema de informação autónomo, replicando dados de outros organismos da mesma organização. Em muitos casos a comunicação de dados é feita de forma artesanal recorrendo a intervenção humana. Estes tipos de transferência de dados são por vezes automatizados mas ficam obsoletos em função das actualizações de cada um dos sistemas de informação. Mesmo conseguida uma comunicação de dados entre os organismos, a validação dos dados raramente é efectuada. A ideia subjacente a este tipo de comunicação é simplesmente transferir dados entre organismos independentemente da forma e circunstancias.

O sucesso da Internet, e em particular do WWW e do HTML, estiveram na base do aparecimento do XML. Esta iniciativa fomentou o aparecimento de várias tecnologias que, em diferentes sectores, vieram contribuir para facilitar a comunicação de dados entre sistemas heterogéneos. Ao nível do desenvolvimento de sistemas de informação surgiram novas tecnologias, nomeadamente linguagens de modelação, que vieram facilitar o desenho e posteriormente a implementação dos mesmos. Este projecto visa desenvolver uma metodologia para a partilha entre sistemas de informação, baseados nestas novas tecnologias, e tendo como motivação resolver os problemas relacionados com a utilização de sistemas heterogéneos.

Na continuação deste são enumeradas características desejáveis nos sistemas de informação. Depois é abordado o problema da partilha de dados em contraste com as bases de dados distribuídas. Seguidamente são enumerados os objectivos pretendidos com este trabalho e a metodologia proposta para esse fim. Por fim é apresentado o exemplo que foi usado na validação desta metodologia.

## 1.1 Sistemas de informação

Nas últimas duas décadas tem sido notória a adaptação das organizações às novas tecnologias e em particular aos sistemas de armazenamento de informação que são normalmente designados por bases de dados. Estes sistemas são conjuntos de informação que descrevem actividades de um ou mais organismos que se relacionam entre si[RG].

Associados aos sistemas de armazenamento estão associados Sistemas Gestores de Bases de Dados (SGBD's), que facilitam a gestão de dados e interacção com aplicações. As principais características desejáveis dum sistema de armazenamento de informação são[For99]:

- **Partilha**. Disponibilizar informação por vários utilizadores e aplicações.
- **Persistência**. Disponibilização permanente de informação.
- **Segurança**. Protecção de informação contra alterações e consultas não autorizadas.
- **Validação**. Verificação da informação face à definição de dados.
- **Consistência**. Consistência das várias instâncias da mesma informação no sistema.
- **Unicidade**. Não devem existir duas instâncias diferentes no sistema com o mesmo significado.
- **Independência**. Os níveis físico e lógico do sistema devem ser independentes dos níveis externos ao sistema.

## 1.2 Partilha de dados entre sistemas heterogéneos

As organizações estão divididas em organismos por vezes dispersos geograficamente. Os organismos podem ser unidades operacionais autónomas, com alguma liberdade de definir as suas próprias regras de funcionamento, incluindo optar pelo sistema de informação e SGBD mais adequado ao seu funcionamento. Desta forma, é frequente que as estruturas de informação e os SGBD's associados aos sistemas de informação de cada organismo sejam diferentes entre si.

### Problema da partilha

Nos organismos pertencentes à mesma organização existe informação comum que é necessária ao seu funcionamento. A informação gerada num organismo tem de ser partilhada pelos organismos que a requerem. Se for usado um sistema de bases de dados distribuídas, a estrutura de informação é a mesma em todos os sistemas de armazenamento e só os dados é que estão distribuídos. No entanto, se os sistemas forem heterogéneos a partilha é dificultada porque o sistema de informação global da organização não tem as características desejáveis, anteriormente enumeradas. Por isso, os organismos acabam por funcionar como ilhas de informação limitando o fluxo de informação.

A figura 1.1 ilustra um cenário do processo de troca de informação entre sistemas de dois organismos. Existem dois sistemas, `A` e `B`, com estruturas de informação e sistemas de armazenamento diferentes. O sistema `A` pretende enviar um conjunto de informação para o sistema `B`. Uma série de passos têm

de ser dados para que a informação enviada pelo organismo A seja interpretada de forma correcta pelo sistema do organismo B.

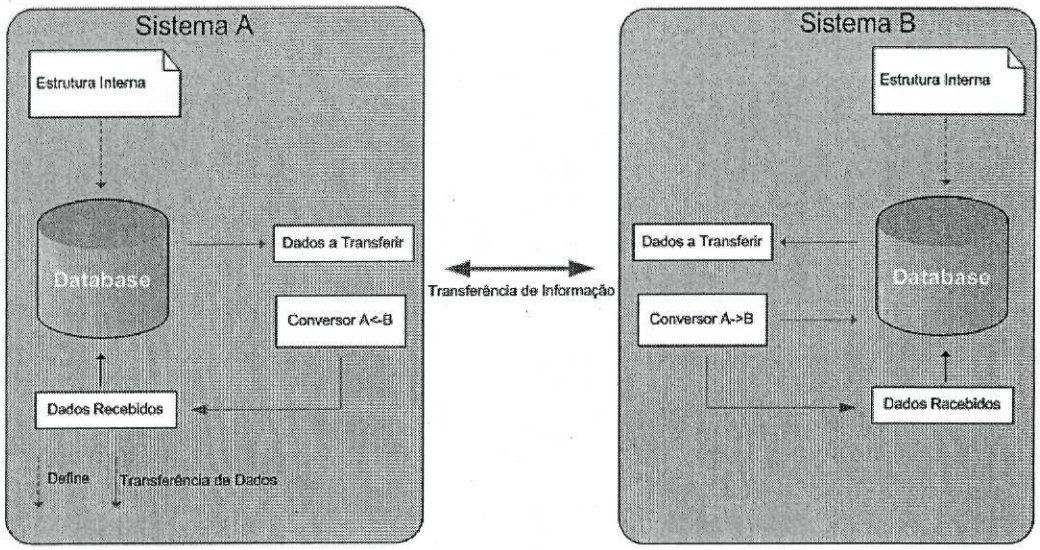



Figura 1.1: Troca de informação entre sistemas heterogéneos.

O primeiro passo do organismo A é obter do seu sistema de armazenamento o conjunto de informação a ser enviado para o organismo B. A informação é enviada de A para B. Como o sistema de armazenamento do organismo B tem uma estrutura de informação diferente da do organismo A, um processo de conversão tem de ser feito para que o sistema de armazenamento do organismo B possa obter a informação no formato pretendido. Após o processo de conversão, o sistema B pode armazenar ou tratar a informação obtida. Um conjunto de passos análogo, embora no sentido inverso, é também necessário para partilhar informação de B para A.

A questão fundamental deste processo é a necessidade de um conversor de informação sempre que a informação é enviada do organismo A para o organismo B e vice-versa. Portanto, sempre que dois sistemas heterogéneos pretendam trocar informação, têm de existir dois conversores, para que ambos a possam interpretar correctamente. Adicionado mais um sub-sistema são necessários quatro novos conversores para que todos os sub-sistemas possam trocar informação correctamente; para N organismos teriam de existir N*N-N conversores. A figura 1.2 representa os 30 conversores necessários numa organização com apenas 6 organismos. Desta forma, à media que se vão adicionando sub-sistemas ao sistema global, o número de conversores aumenta quadraticamente tornando este método de transferência de informação ineficiente, e impraticável nos casos em que o número de sub-sistemas é elevado.

## 1.3 Metodologia para partilha de dados

### 1.3.1 Objectivos

De forma geral o objectivo deste trabalho é definir uma metodologia para resolver o problema da partilha de dados entre sistemas de informação heterogéneos de diferentes organismos. Quando se está a falar de partilha no âmbito deste projecto, apenas se pretende comunicar dados entre os diferentes organismos.

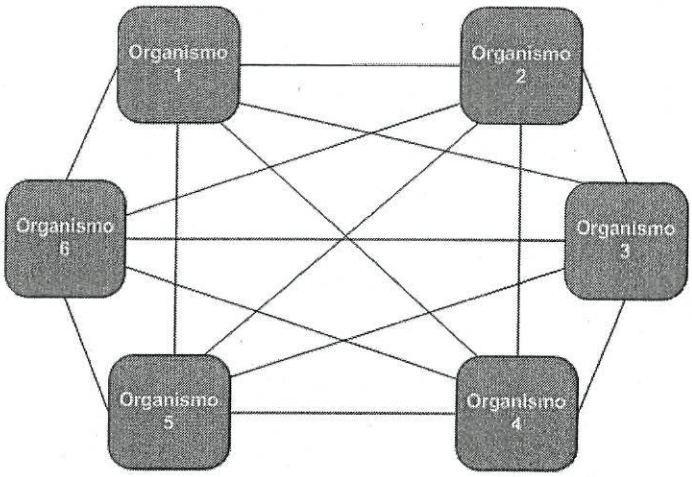



Figura 1.2: Conversores numa organização com 6 organismos.

De uma forma mais detalhada, os objectivos que nos propomos atingir são os seguintes:

- Criar uma especificação do sistema global (funcionamento da organização). Reunindo toda a informação relevante dos sistemas de informação dos organismos. Deve apresentar a informação que cada organismo pode disponibilizar e a informação que necessita de outros organismos. Desta forma definem-se relacionamentos entre os organismos, isto é, um processo de partilha.
- Definir a atribuição de responsabilidades aos organismos que têm de disponibilizar informação e conceder permissões às que a requerem.
- Definir uma representação da informação a partilhar para normalizar o formato da mesma.
- Definir um modelo de comunicação para a partilha de dados. Este modelo deve tirar partido do formato de dados obtido anteriormente. Os objectivos deste modelo de comunicação é uniformizar a forma dos organismos comunicarem e também restringir o formato dos dados transmitidos através da definição de dados.

### 1.3.2 Abordagem

O desenvolvimento da metodologia foi feito em três etapas, para tentar cumprir os objectivos enumerados. As etapas da metodologia seguem uma ordem, o que não obriga a que quando se concluí uma etapa não se possa voltar a trás para fazer algumas alterações. Sempre que se justifiquem alterações em etapas anteriores para benefício da metodologia global, isso deve ser feito. As etapas são as seguintes:

- A primeira etapa é obter uma representação do sistema de informação a partilhar na organização. Para isso é reúnem-se os organismos e é feita uma recolha da informação que cada um gera. Depois cada organismo identifica a informação que necessita de obter dos outros. Definidos os relacionamentos de partilha entre as entidades, são atribuídas responsabilidades aos organismos que devem disponibilizar informação e conceder direitos de acesso aos organismos que a requerem.

Depois de identificada a informação para a partilha entre os organismos, deve-se definir o tipo de dados da mesma para que todos os organismos a cumpram. Isto adiciona mais um nível de validação da informação partilhada entre os organismos, porque a informação comunicada tem de confirmar com a sua definição.

Para concretizar esta etapa devem ser anotadas todas as decisões tomadas. Para isso deve ser definido um modelo de dados que ilustre a partilha de informação. Para finalizar esta etapa, o modelo de dados deve ser representado numa linguagem de modelação para que todos os organismos intervenientes na partilha tenham acesso ao modelo estabelecido e tenham conhecimento da definição de dados que têm de respeitar na partilha de dados.

- A segunda etapa é a obtenção de uma definição de dados independente do organismo. Isto é, uma definição de dados única para todos os organismos que deve ser usada como formato de dados na partilha de informação. A definição de dados que se pretende obter nesta etapa resulta de um processo de mapeamento do modelo de dados definido na etapa anterior.

  Devido ao facto da definição de dados obtida englobar definições usadas por todas as partilhas de dados entre os organismos achamos que seria conveniente restringir essas definições a cada organismo gerando uma definição por organismo. Passa a existir uma definição de dados única por organismo apenas com as definição do seu âmbito.

  Como nesta etapa apenas se criam definições de dados para os organismos num formato independente, seria também interessante ter a possibilidade de poder tirar partido dos mesmos para definir outro tipo de definições de dados. Isto é, a possibilidade de gerar definições de dados especificas a alguns sistemas de armazenamento a partir das definições obtidas.

  Esta solução diminuí a ordem de grandeza do número de conversores necessários para 2N. A figura 1.3 representa este novo processo de troca de informação.

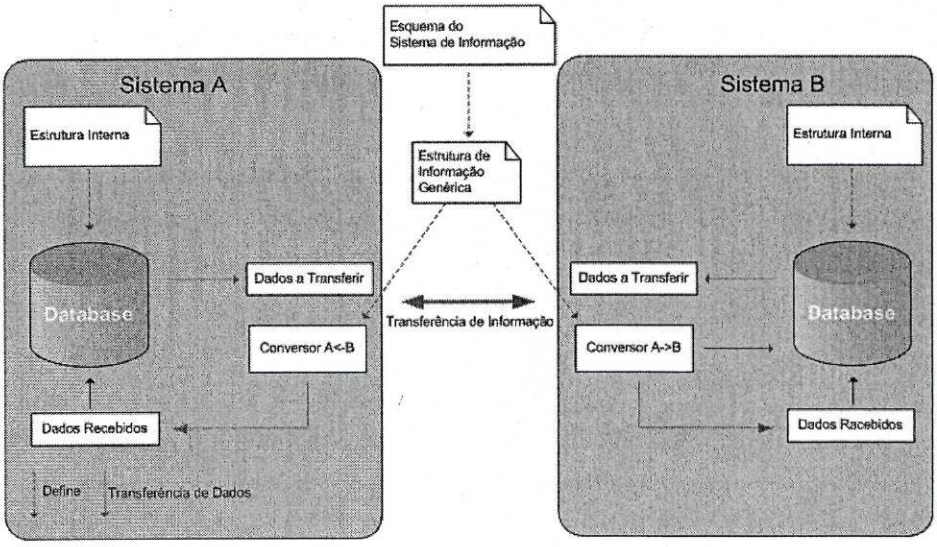



Figura 1.3: Nova troca de informação entre dois sistemas heterogéneos.

- Por último definiu-se um modelo de comunicação entre os organismos usando as definições de dados obtidas na etapa anterior. O modelo de comunicação deve determinar uma forma de disponibilização de dados e de acesso por parte dos organismos envolvidos na partilha. Resumindo, os organismos fazem pedidos de informação a outros, e estes respondem aos pedidos com a informação desejada. O modelo de comunicação também permite o uso de recursos de

segurança que garantam a identidade dos organismos envolvidos, a integridade, privacidade e autenticidade da informação transmitida.

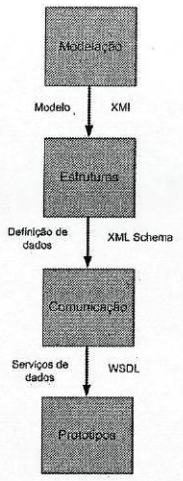



Figura 1.4: Abordagem da metodologia.

Existem várias abordagens para resolver este problema. No âmbito deste projecto procura-se uma abordagem genérica, o mais independente possível das plataformas e ferramentas de desenvolvimento. Mais concretamente, pretendemos definir uma metodologia que deixe em aberto opções como as ferramentas e plataformas usadas na sua concretização. Por este motivo a metodologia que propomos baseia-se em *standards*, nomeadamente o UML e o XML. Esta metodologia não faz qualquer comparação de ferramentas nas suas etapas. AS ferramentas apresentadas foram propositadamente escolhidas para serem multi-plataforma e de acesso gratuito, permitindo minimizar custos. No caso de não existirem limitações orçamentais o uso de algumas ferramentas comerciais podem trazer algumas vantagens ao nível da implementação e desempenho da solução.

### 1.3.3 Exemplo

O desenvolvimento da metologia foi acompanhado por um exemplo da aplicação da mesma sobre um cenário. Este exemplo foi usado principalmente para verificar a correcta definição da metodologia à medida que ia sendo definida, ajudando a encontrar problemas nas várias etapas, fazer alterações em etapas anteriores sempre que se verificassem problemas daí resultantes ou caso se considera-se vantajoso e visualizar a aplicação da metodologia que facilita bastante o processo de desenvolvimento.

O cenário escolhido para usar no nosso exemplo foi o de uma Faculdade, que é uma organização dividida em organismos autónomos, como os departamentos e os serviços centrais (académico, financeiro, patrimonial, científico, etc), que necessitam de trocar informação .

Os organismos que compõem uma Faculdade podem estar geograficamente distribuídos. Têm alguma autonomia de funcionamento, isto é, podem escolher o sistema de informação informação que melhor se adeqúe ao seu funcionamento implementando-o sobre uma qualquer plataforma ou sistema. No entanto, têm de cumprir determinadas regras impostas pelo Conselho Directivo da Faculdade que funciona como uma entidade reguladora.

Aumentando o nível do cenário, ao examinarmos o trajecto da informação referente a um aluno que entra para o Ensino Superior, desde o dia em que o aluno se candidata, até ao dia em que tem a

primeira aula prática, num Departamento da Faculdade a que o seu curso está associado, é visível a necessidade dos organismos do Ensino Superior comunicarem informação para poderem funcionar.

Os organismos necessitam de informações de outros organismos pertencentes à mesma organização. Um departamento de uma Faculdade necessita de informação relativa aos alunos e disciplinas fornecida pela secretaria da Faculdade. Por sua vez a secretaria necessita de informação relativa aos novos alunos da Faculdade que obtém do Ministério da Educação. O Ministério da Educação necessita de informação dos candidatos ao preenchimento das vagas para o Ensino Superior que é fornecida pelas Delegações Regionais onde os alunos se candidatam. A figura 1.5 apresenta a forma como estão organizados os vários organismos pertencentes ao Ministério da Educação[1].

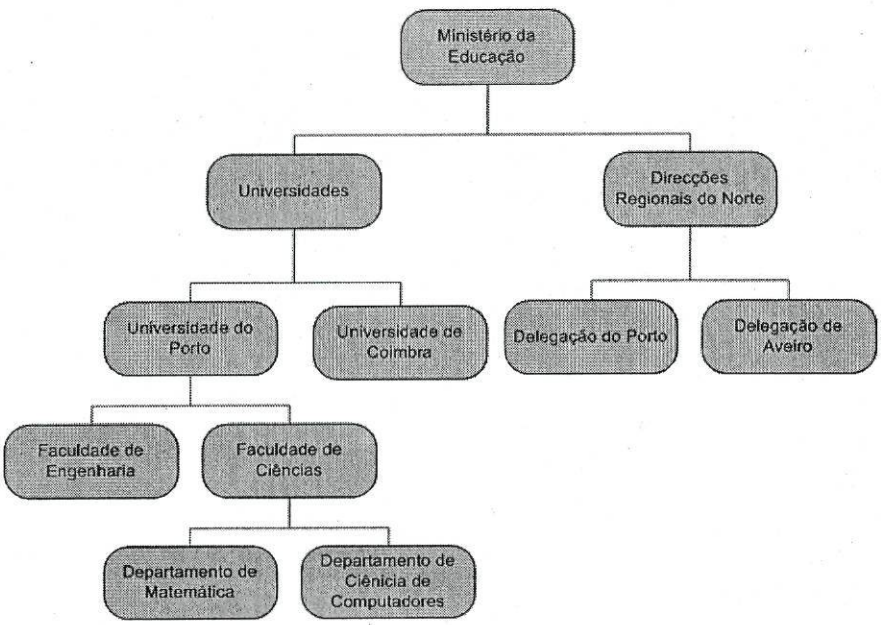



Figura 1.5: Possível organização do Ministério da Educação.

Particularizando a organização Faculdade, existe informação que pertence ao domínio comum dos organismos (departamentos, secretaria, etc). A informação mais evidente é a relativa aos alunos. Quando um aluno é matriculado numa Faculdade, os departamentos em que o aluno irá ter aulas necessitam não só de obter informação relativa aos seus dados pessoais, bem como as disciplinas a que está inscrito. No inicio de cada ano lectivo a secretaria da Faculdade tem a responsabilidade de fornecer este tipo de informação aos departamentos. No fim de cada semestre, após a avaliação ter terminado, os resultados que os alunos obtiveram às disciplinas leccionadas nos departamentos têm de ser enviados para a secretaria da Faculdade, pois esta é a responsável por toda a informação relativa aos alunos e disciplinas.

## 1.4 Trabalho relacionado

Para estabelecer a metodologia que vai ser apresentada foi feito um trabalho de pesquisa de trabalhos relacionados com a nossa problemática. Por ser um tema bastante actual não foi encontrado nenhum trabalho que apresente uma solução para os objectivos enumerados. Existem diversos trabalhos que focam os objectivos mas individualmente. Alguns desses trabalhos são apresentados em seguida,

[1] A organização do Ministério de Educação apresentada é fictícia

estando os primeiros relacionados com representação e conversão de dados e os segundos com os serviços de dados[2].

### 1.4.1 Conversão de dados

#### UML como esquema à partilha de dados em XML

O UML (*Unified Modeling Language*) é uma linguagem de modelação eficaz e flexível com capacidades de descrever aspectos estáticos, dinâmicos e comportamentais de sistemas de informação. Têm sido feitos alguns estudos com o objectivo de usar o UML para definir um esquema de partilha de dados em XML.

Num estudo feito por *David Skogan*[Sko99]. A conclusão é que o UML com algumas restrições pode ser usado para definir a estrutura e semântica dos dados. Por seu lado, os dados em formato XML é o adequado para permitir interoperabilidade. A ideia geral é usar a representação dos diagramas UML segundo a especificação XMI (*XML Metadata Interchange*), que só representa aspectos estáticos dos modelos UML. Depois, segundo um processo de mapeamento gerar definições de dados XML (DTD, *Schema*). Desta forma fica definido um formato de dados XML para troca de informação.

Existem vários projectos que definiram este processo de mapeamento de modelos UML para definições de dados XML como: *DISGIS European ESPIRIT project n. 22.084*, o *ISO Technical Committee n. 211 Geographic Information* e *ISO 15046*.

#### Combinar UML, XML e bases de dados relacionais

O projecto *FieldWorks* da empresa de desenvolvimento de software *SIL International's* é um conjunto de aplicações para investigação linguística e antropológica. O desenvolvimento aborda aspecto como modelação, armazenamento e técnicas de obtenção de dados[HH01].

É feita uma análise orientada a objectos para criar os modelos, posteriormente representados em UML. Os modelos de dados são depois convertido para a especificação XMI. Partindo desta representação em XML, as suas equipas de trabalho podem obter outras representações dessa informação usando folhas de estilo XSL. Dessas representações fazem parte: um DTD para validação dos dados de importação, documentação em HTML dos diagramas de classes UML e um esquema de uma base de dados relacional. Resumindo, este projecto tira partido da maturidade das bases de dados relacionais para o armazenamento de dados e do XML para partilha de dados.

Usam o *SQL Server 2000* como sistemas de armazenamento de dados, que dispõe de uma ferramenta para relacionar elementos de um documento XML com a estrutura interna da base de dados. Desta forma a troca de dados é sempre feita segundo o mesmo esquema, porque existe sempre um mapeamento entre o formato independente e o formato do sistema de armazenamento.

[2]As tecnologias enumeradas em seguida serão descritas no Capítulo2.

### DB2XML

O DB2XML é uma ferramenta para passar informação de bases de dados relacionais para documentos XML. É uma ferramenta Java que comunica com as bases de dados através de JDBC. Pode funcionar de forma independente ou como um *servlet* em aplicações Web[Tur99]. As suas principais funcionalidades são:

- Transformar o resultado de consultas a bases de dados ou a base de dados total em documentos XML.
- Geração de definições de dados (DTD) que caracterizam os documentos XML obtidos.
- Transformação dos documentos XML obtidos usando folhas de estilo XSLT.

Os principais motivos que levaram à criação de uma ferramenta como esta são:

- **Melhor desenho.** Esta solução permite separar aspectos do armazenamento da apresentação de dados.
- **Suporta diferente clientes.** A utilização de folhas de estilo permite apresentar os dados resultantes em diferentes formatos.
- **Distribuição da computação.** A computação passa a estar distribuída pelo cliente e pelo servidor.
- **Menor largura de banda.** Os requisitos ao nível da largura de banda para documentos HTML são normalmente superiores aos necessários para documentos XML e folhas de estilo juntos.

### XML-DBMS

O XML-DBMS é uma ferramenta intermédia para transferência de dados entre documentos XML e bases de dados relacionais[Bou]. Mapeia os documentos XML para a base de dados de acordo com um mapeamento do tipo objecto-relacional em que normalmente os elementos são vistos como classes e os atributos como propriedades das classes.

Define uma linguagem de mapeamento baseada em XML que permite ao utilizador especificar e caracterizar o mapeamento. Esta linguagem define o mapeamento da estrutura de um documento XML para a estrutura de uma base de dados e vice-versa.

É uma ferramenta disponível em Java e em Perl que pode ser usada na linha de comando ou então como parte de uma aplicação. O seu código fonte está disponível para qualquer fim.

## 1.4.2 Serviços de dados

Os sistemas para comércio electrónico B2B (*Business-to-Business*) são também designados por servidores de negócio e funcionam como intermediários entre aplicações servidor e os clientes, proporcionando serviços de transferência e tradução de mensagens. O ênfase deste tipo de serviços é permitir a comunicação entre aplicações numa organização[Bea01].

Existem três requisitos essenciais que caracterizam um sistema B2B:

- Representação de dados.
- Transferência de mensagens.
- Modelar o processo de negócio.

Todos os servidores B2B cobrem os dois primeiros requisitos, enquanto que o terceiro é considerado por apenas alguns. É um requisito que está longe de obter consenso por parte sistemas B2B do mercado.

As soluções para responder aos requisitos apresentados variam consoante o sistema usado, contudo, as soluções mais usadas são as seguintes:

- XML para representação de dados.
- Uso de protocolos Internet para a comunicação.
- Sistemas específicos para o processo do negócio.

A definição de sistemas para o processo de negócio é um problema complicado de solucionar. Por um lado pretende-se ter uma completa flexibilidade para que o negócio seja efectuado por um protocolo qualquer para o efeito, por outro lado pretende-se que os organismos comuniquem segundo *standards* em que o nível do coordenação na comunicação seja mínimo. Este é o grande problema dos sistemas actuais, uns deixam em aberto a escolha do processo de negócio, enquanto que outros estabelecem consórcios para definir *standards* para este processo.

Em seguida são apresentadas as principais organizações com sistemas B2B:

- **BizTalk Framework.** Fundada e suportada pela Microsoft, o BizTalk é um sistema de transferência de mensagens baseado em XML (http://www.biztalk.org).
- **RosettaNet.** Consorcio de empresas na área de implementação de sistemas para comércio electrónico. Define formatos e processos específicos de negócio (http://www.rosettanet.org).
- **ebXML.** Iniciativa conjunta da OASIS (*Organization for the Advancement of Structured Information Standards*) e da UN/CEFACT (corpo das Nações Unidas dedicada ao desenvolvimento do comércio electrónico) para desenvolver formatos e processos de B2B (http://www.ebxml.org).

# Capítulo 2

# Estado da Arte

Para poder cumprir o principal objectivo desta metodologia, que é permitir a comunicação de dados entre sistemas de informação heterogéneos, foram usados *standards*. Esta decisão foi tomada com vista a normalizar o mais possível todo o processo de definição e comunicação de dados entre os organismos.

Este capítulo descreve os *standards* que foram utilizados no projecto. Começa por apresentar a modelação de dados no desenvolvimento de aplicações abordando a linguagem UML criada pela organização OMG. Depois apresenta o W3C, a organização responsável pelos estandardização das tecnologias. Depois é apresentada a especificação XML, enumerando os princípios que o caracterizam e as linguagens que descrevem e validam documentos XML. De seguida são apresentadas duas convenções XML, os *Namespaces* e o *XPath*. Depois são descritos alguns dos *standards* XML: XSL e XSLT, XQuery e os *Schemas*. Aos últimos é dada especial atenção. O processamento de documentos XML, através do DOM e do SAX, é outro dos pontos apresentados. As aplicações do XML é tema abordado logo a seguir. O último ponto relativo ao XML é a apresentação de três aplicações XML: XHTML, SVG e XMI.

## 2.1 UML

O *Unified Modeling Language* (UML) é uma linguagem gráfica *standard* para criar modelos de sistemas de software desenvolvida pela OMG (*Object Management Group*), organização fundada em 1989 pela união de onze principais companhias de tecnologias de informação. O objectivo desta organização é estabelecer especificações na industria de desenvolvimento de aplicações baseadas em objectos.

As tecnologias de modelação orientadas a objectos, ganharam muitos adeptos. Contudo, muitos deles consideravam difícil a tarefa de encontrar uma linguagem de modelação que correspondesse completamente às suas necessidades. Alguns investigadores nesta área começaram por disponibilizar algumas ideias de linguagens. Mais tarde, um trabalho conjunto de alguns dos principais investigadores nesta área, originou o UML, como uma unificação dos processos de modelação. Esta passou a ser a linguagem adoptada para o desenho de modelos de sistemas de informação. Como esta linguagem surge da reunião de diversos métodos e ideias, consegue abranger múltiplas visões da modelação, obtendo uma aceitação quase automática.

O UML pode ser usado para visualizar, especificar, construir e documentar as características mais relevantes dos sistemas de informação[BRJ99, Alh98]. O UML define modelos que abrangem várias áreas, desde modelos de requisitos funcionais e percursos dos fluxos de dados, até desenhos de estruturas de classes e diagramas de componentes. Desta forma, é possível obter várias visões de um mesmo sistema. A utilização do UML facilita e simplifica o processo de desenho dos sistemas, já que é usada uma única linguagem por todos participantes neste processo.

A sua expressividade facilita o processo de modelação, bastando para isso conhecer o modelo conceptual da linguagem, que é composto por três elementos principais: os blocos básicos do UML, as regras que ditam como é que os blocos se podem conjugar, e alguns mecanismos que são aplicados em toda a linguagem.

O UML é apenas uma linguagem de modelação independente do processo de desenvolvimento, mas para se tirar o máximo partido dos modelos, deve ser usado num processo de desenvolvimento pratico, iterativo e incremental.

### A Linguagem

O UML é uma linguagem que proporciona um vocabulário e um conjunto de regras para combinar palavras desse vocabulário. Como linguagem de modelação o vocabulário e a regras focam o a representação física e conceptual do sistema. O UML é uma linguagem de:

- **Visualização**. Para muitos programadores, a distância entre idealizar uma implementação e depois passar à prática (escrita de código), é praticamente zero. Imaginam, e escrevem. De facto, para algumas coisas é preferível escrever o código directamente. Nestes casos, o programador faz alguma modelação, mas apenas mentalmente. Possivelmente até expressa algumas ideias através de esboços. Contudo, existem alguns problemas com desenvolvimentos desta natureza. O primeiro é a comunicação desses modelos conceptuais à equipa de desenvolvimento. Toda a equipa tem de utilizar o mesmo modelo, caso contrário, poderão existir graves problemas de comunicação. O segundo problema é a existência de determinadas características dos sistemas de software que só podem ser descritas através de modelos. O terceiro problema é o facto do programador nunca documentar o modelo de programação que utilizou. Essa informação, é perdida para sempre, ou no melhor caso, alguma dessa informação é recuperada a partir da implementação.

  O UML é mais do que um conjunto de símbolos gráficos. Por detrás de cada símbolo existe uma semântica bem definida. Desta forma, um programador pode escrever um modelo UML, e outro programador ou ferramenta pode interpretar esse modelo. Isto resolve o primeiro problema.

  Algumas coisas são preferencialmente modeladas textualmente; outras graficamente. Em todos os sistemas, existem estruturas que ultrapassam o limite da possibilidade de representação através de uma linguagem de programação. O UML permite as duas visões, resolvendo assim o segundo problema.

  Escrever modelos em UML soluciona o terceiro problema, ou seja, criar um modelo explicito facilita a comunicação.

- **Especificação**. Neste contexto, especificar significa construir modelos precisos, inequívocos e completos. Em particular, o UML apresenta uma especificação de todas as análises importantes,

desenhos e decisões de implementação que devem ser tomadas durante o desenvolvimento e disponibilização de sistemas de software.

- **Construção**. O UML não é uma linguagem de programação gráfica, mas os modelos podem estar directamente ligados a uma variedade de linguagens de programação. Isto quer dizer que é possível mapear modelos UML para linguagens de programação como o Java, C++, Visual Basic, ou até para tabelas de bases de dados relacionais.

  Este mapeamento permite *forward* e *reverse engineering*. *Forward engineering* é a actividade no processo de desenvolvimento em que a informação de saída de uma aplicação que é usada como entrada noutra aplicação. *Reverse engineering* é a actividade de análise, compreensão e representação de sistemas já implementados. Combinando estes dois passos, permite *round-trip engineering*, que significa a possibilidade de trabalhar sobre uma vista gráfica ao mesmo tempo que uma vista textual, enquanto que as ferramentas mantêm ambas as visões consistentes.

- **Documentação**. Uma organização de desenvolvimento de software produz todo o tipo de artefactos (planos, protótipos, desenhos,etc) juntamente com o código. Dependendo da cultura de desenvolvimento, alguns dos artefactos são tratados de uma maneira mais formal do que outros. Estes artefactos não são só importantes quando o sistema é disponibilizado, mas também no controlo, medição e comunicação durante processo de desenvolvimento e depois de publicado.

  O UML permite documentar todos os detalhes da arquitectura de um sistema.

O vocabulário e as regras do UML indicam a forma de criar modelos, mas não indicam que modelos é que se devem fazer, nem quando os fazer. Isso cabe ao processo de desenvolvimento definir.

### 2.1.1 Modelação

O sucesso do desenvolvimento de software assenta na qualidade e na resposta às necessidades dos utilizadores. Uma instituição nesta área, que cumpra os objectivos de forma atempada e previsível, com uma gestão eficiente dos recursos físicos e materiais, tem todas as condições para singrar no mercado[BRJ99].

Para disponibilizar software que cumpra os objectivos propostos, é necessário confrontar os utilizadores de forma a exponham as necessidades reais do sistema. Para desenvolver software persistente, de qualidade, é necessária capacidade de criar uma arquitectura sólida susceptível a alterações. Para desenvolver software da forma mais rápida, eficiente e efectiva, com o mínimo de reescrita e fragmentos, com a noção dos custos do tempo de vida do sistema, é necessário ter um processo de desenvolvimento que se possa adaptar à alteração dos negócios e das tecnologias.

A modelação é uma actividade que pode contribuir para o bom desenvolvimento de software. Os modelos existem para:

- Representar a estrutura e comportamento de um sistema.
- Visualizar e controlar a arquitectura dos sistemas.
- Compreender melhor o sistema que se está a construir.
- Gerir o processo de desenvolvimento.

### A importância de modelar

Com o aparecimento da Internet, a necessidade de desenvolvimento de software aumentou, e as equipas de desenvolvimento tendem a fazer aquilo que realmente sabem, escrever linhas de código. Hoje em dia o importante é programar o mais possível para responder a mais necessidades, em vez de se programar da forma mais eficiente. Contudo, esta prática pode não originar as linhas de código mais correctas, e alguns projectos têm uma magnitude tal, que mesmo que o número de horas de trabalho aumente, o trabalho pode não ser concluído correctamente.

A grande dificuldade no desenvolvimento de software é a criação de software o mais correcto possível, encurtando ao máximo a sua escrita. Isto caracteriza o software resultante como descendente directo da arquitectura, do processo e das ferramentas de desenvolvimento.

Enquanto que o insucesso de alguns projectos de software deve-se a falhas próprias, a maioria dos projectos de sucesso têm alguns pontos em comum. O elemento, mais comum, que origina o sucesso de uma instituição de desenvolvimento de software é a modelação.

### O que é um modelo?

Um modelo é um esquema ou planificação de um sistema. Um modelo pode incorporar diversas visões detalhadas, do sistema em consideração. Um bom modelo incluí os elementos que têm grande importância, e omite os elementos irrelevantes a um determinado nível de abstracção. Cada sistema, pode ser descrito de diferentes formas, através de diferentes modelos. Um modelo pode ser estrutural, realçando a organização do sistema, ou pode ser comportamental, realçando os dinamismo do sistema.

### Porquê modelar?

Existem quatro razões fundamentais para modelar:

- Os modelos ajudam a visualizar o sistema como ele é, ou como se pretenda que seja.
- Os modelos permitem especificar a estrutura e o comportamento de um sistema.
- Os modelos definem uma matriz que orienta a construção do sistema.
- Os modelos documentam as decisões que se tomam.

A modelação não é só utilizada para caracterizar grandes sistemas. Pode ser utilizada em sistemas de qualquer dimensão. Contudo, é verdade que para sistemas maiores e mais complexos, a modelação tem um papel mais preponderante, por uma simples razão: a capacidade humana de compreensão da complexidade é limitada. Através da modelação, a atenção é centrada apenas num aspecto de cada vez. Isto é essencialmente a abordagem "dividir para conquistar". A modelação permite ainda ampliar a inteligência humana, permitindo ao modelador trabalhar a níveis de abstracção mais elevados.

Dizer que existem ocasiões em que não é necessário um modelo, não é propriamente verdade. Na realidade, até para o mais pequeno projecto, é feita modelação, embora podendo ser feita

informalmente. O programador pode esboçar uma ideia no papel ou num quadro, para visualizar uma parte do sistema, ou até representar um exemplo do funcionamento. Tudo isto são modelos. Contudo, este modelos informais são considerados *ad hoc*, não proporcionando uma linguagem comum que possa ser partilhada por outros programadores.

Qualquer projecto beneficia do uso de modelos. Mesmo em alguns casos, em que se concluí que existem componentes de software desnecessárias, os modelos podem ajudar a visualizar e planear um desenvolvimento mais rápido e correcto. Quanto mais complexo for o projecto, maior é a probabilidade criar coisas errados, no caso de não se usar modelos.

### Os princípios da modelação

A experiência no uso da modelação identificou quatro princípios fundamentais:

- **A influência do modelo na implementação.** A escolha dos modelos a utilizar tem de ser bem feita. Os modelos correctos ilustram correctamente os problemas de desenvolvimento mais minuciosos; os modelos errados desviam as atenções para problemas de menor relevância.

  Em programação, os modelos que se escolhem têm grande influência na visão dos projectos. Se um sistema for construído por um desenhador de bases de dados, decerto que se vai focar em modelos entidade-relacionamento. Se um sistema for construído por um analista de estruturas, é provável que os modelos utilizados sejam centrados em algoritmos. Se um sistema for construído por um programador de aplicações orientadas a objectos, a arquitectura implementada assenta num conjunto de classes. Qualquer uma destas aproximações pode estar correcta para uma determinada aplicação e cultura de desenvolvimento. A conclusão a que se chega é que cada visão torna o sistema diferente, com diferentes custos e benefícios.

- **Cada modelo deve ser expresso com vários níveis de precisão.** Por vezes, uma simples e rápida elaboração de um modelo é exactamente o necessário; noutras ocasiões, é necessário descer a um nível mais pormenorizado. Em qualquer dos casos, os melhores modelos são aqueles que permitem escolher o grau de detalhe, dependendo de quem está a criar a visão e qual o motivo da visão. Um analista ou um utilizador necessita de uma visão do sistema diferente de um programador.

- **Os melhores modelos são os relativos a implementações reais.** Os modelos devem ter uma clara ligação com a realidade, e nos casos em que existe uma ligação mais fraca, saber exactamente como é que os modelos divergem do mundo real. Todos os modelos devem simplificar a realidade; o truque é ter a certeza que algumas simplificações não escondam alguns detalhes importantes.

- **Um único modelo é insuficiente.** Para se entender um sistema de software orientado a objectos, são necessárias várias visões complementares e inter-ligadas: visão de desenho, visão do uso, visão de processos, visão de implementação, etc. Cada uma destas visões podem representar aspectos estruturais, como comportamentais. O conjunto de todas as visões representam o projecto de software.

### Processo de modelação

O processo de modelação é totalmente independente da tecnologia de implementação do projecto. Este processo pode envolver dois tipos de modelos [Bea01]:

- **Modelo estático**. Representação dos objectos que estão envolvidos no domínio do problema - é necessário dar-lhes um nome e descreve-los.
- **Modelo dinâmico**. Representação de como a informação é gerada e deslocada de um local para outro, até a tarefa ser terminada. Pode envolver diagramas que representem os processo e fluxos de informação e os ciclos de vida dos objectos.

Ambos os modelos, necessitam de descrever:

- Os diferentes estados em que o sistema pode estar
- Propriedades
- Relacionamentos

No processo de modelação podem ser usadas linguagens especificas, como o UML, mas deve-se usar princípios mais genéricos para poder atingir os seguintes objectivos: obter uma definição precisa dos conceitos modelados e comunicar o modelo a todos os utilizadores.

Enquanto que as linguagens oferecem maior precisão na definição dos modelos, é necessário ter conhecimento da mesma. Caso algum utilizador não o tenha, não poderá entender a definição.

Independentemente do tipo de modelo pelo qual se comece a definição, é importante: não passar à fase seguinte sem acabar a anterior e existir sempre a possibilidade de voltar a fases anteriores para fazer alterações.

O processo de qualquer um dos tipos de modelos segue os passos seguinte:

1. Identificar, nomear e definir objectos
2. Definir e refinar relacionamentos
3. Adicionar propriedades aos objectos

### Identificar, nomear e definir objectos

Este primeiro passo é apenas de escrita de todas as componentes de um sistema. Por exemplo, clientes, empregados, departamentos, produtos, etc. Não é necessário perder muito tempo neste passo, já que se pode sempre voltar este para acrescentar coisas que se considerem relevantes, durante outros passos do processo de modelação.

As componentes designam-se por tipos de objectos. Depois de feita a identificação destes, é possível defini-los. A definição é importante porque elimina ambiguidades. Existem muitas perguntas e afirmações que se podem fazer para ajudar a obter definições correctas, como por exemplo:

- X é uma encomenda?

- X e Y são a mesma coisa?

Se estas questões foram feitas a diferentes pessoas, é provável que se obtenham diferentes respostas, que apenas serve para realçar a importância do processo de modelação. É necessário garantir que todos os participantes na criação e utilização do sistema o entendam perfeitamente.

É também relevante a forma como são identificadas as instâncias dos objectos. Já existem identificadores industriais únicos para alguns tipos de objecto, contudo, existem outros que tem de se encontrar uma forma de identificar unicamente.

No fim deste passo é visível uma lista de objectos e as suas definições, mas convém lembrar que existe sempre a possibilidade voltar a este passo para adicionar e alterar novos elementos.

### Definir e refinar relacionamentos

Depois de obtidos os objectos é necessário definir os relacionamentos existentes entre eles. Após identificados, os relacionamentos podem ser especializados, de forma a caracteriza-los o mais possível.

A caracterização dos relacionamentos pode incluir refinamentos. Os mais usados são a cardinalidade e as restrições. A cardinalidade permite indicar a participação de cada objecto no relacionamento. Esta participação pode ser definida por um único valor ou por um intervalo de valores. As restrições podem ser adicionadas através de caracterizações extra do tipo de relacionamento, como as agregações e as composições.

### Adicionar propriedades aos objectos

As propriedades servem para refinar os objectos. Normalmente, representam conceitos que não existem fora do âmbito dos objectos. São usadas para definir os tipos de dados, os valores que são permitidos, as unidades de medida, valores por defeito e identificar componentes opcionais.

É importante salientar o facto da modelação estar um nível acima das tecnologias de implementação - as propriedades são adicionadas apenas no modelo de dados abstracto. A implementação dessas propriedades depende muito da tecnologia que é usada. Por exemplo, comparando duas implementações do mesmo modelo, quem estivem familiarizado com bases de dados relacionais irá provavelmente criar tabelas em que as propriedades dos objectos apenas têm um papel (por exemplo, apenas é representado o preço, não o tipo de moeda); outra forma de o fazer é normalizar o modelo entidade-relacionamento, originando outras tabelas. Se a tecnologia utilizada for o XML este problema já não acontece, pois considerando um objecto como um elemento, ele pode conter várias propriedades representadas como atributos ou elementos.

No fim deste passo, é obtido um modelo estático completo, com as definição das classes de objectos, apresentadas pelo sistema, as relações entre elas, e as suas propriedades.

### O papel da modelação neste trabalho

A modelação como parte fundamental para o bom desenvolvimento de um projecto de software, é o primeiro passo que este projecto de tese descreve. As razões que justificam o uso da modelação já foram apresentadas na secção anterior, contudo, existem alguns pontos importantes que têm de ser salientados. Este pontos são fundamentais para cumprir os objectivos que se pretende com o projecto.

- **Obter estruturas de dados.** No fim do processo de modelação têm de ser obtidas estruturas de dados que possam ser usadas na implementação do sistema. Neste projecto em particular, as estruturas de dados vão servir de esquema de representação da informação que é partilhada pelos organismos. No âmbito deste projecto, um organismo significa uma componente (física ou abstracta) que participa na troca de informação. Por exemplo, no cenário apresentado, a Secretaria é um organismo, pois gera informação sobre os Alunos, que é enviada para os Departamentos, que são também organismos.

- **Identificar organismos.** A primeira fase desse processo é a identificação dos objectos do sistema. Para este projecto esta identificação é insuficiente, pois é necessário determinar quais os organismos participantes no sistema. Para isso é necessário elevar o processo de modelação a uma nível acima.

- **A atribuição de responsabilidades e permissões dos organismos.** Na identificação dos organismos tem de estar bem definido o papel de cada um, isto é, associado a cada organismos deve estar um conjunto de responsabilidades e permissões. As responsabilidades dizem respeito ao conjunto de informação que estão responsabilizados por gerir. Segundo o cenário apresentado, a Secretaria tem a responsabilidade de gerir toda a informação relativa aos alunos, por outro lado, são os Departamentos que têm a responsabilidade de gerir as notas dos Alunos às Disciplinas. A responsabilidade não se refere apenas à necessidade de uma boa gestão da informação, os organismos devem manter a informação disponível para que os organismos que necessitem da mesma a possam obter. As permissões dos organismos referem-se à possibilidade de obter determinada informação de um organismo. Voltando ao exemplo, o Departamento de Pessoal não deve poder obter informação relativa aos Alunos. Cabe a cada organismo manter os sistemas de disponibilização de informação devidamente seguros, para que só possam obter informação os organismos que tiverem permissões para tal.

## 2.1.2 Modelo conceptual

A criação de modelos em UML é feita segundo um conjunto de diagramas pertencentes às componentes do seu modelo conceptual. Do seu modelo conceptual fazem parte componentes como regras e mecanismos de consistência. Qualquer modelo definido em UML tem por base um dos nove tipos de diagramas, tendo cada um deles um propósito bem definido. Um diagrama UML pode ser considerado um grafo tipado. Isto é, existe um conjunto pré-definido de nós, arcos e regras. Os nós correspondem às componentes, os arcos aos relacionamentos e a regras definem a forma como podem ser ligados os nós através dos arcos, dependendo do diagrama. A figura 2.1 apresenta um diagrama de classes UML e um grafo em que são visíveis as semelhanças entre ambos.

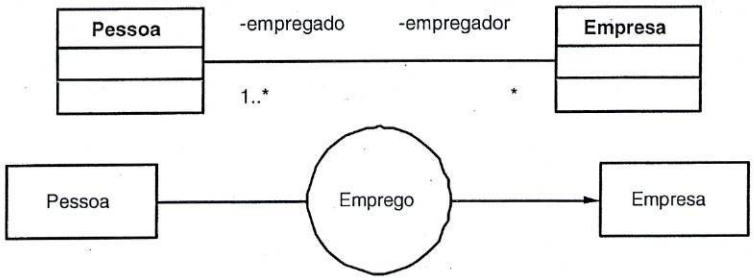



Figura 2.1: Semelhanças entre um diagrama de classes UML e um grafo.

Em seguida são apresentadas as três partes do modelo conceptual do UML: as componentes, as regras e os mecanismos de consistência.

### Componentes

O vocabulário do UML incorpora três tipos de componentes: elementos, relações e diagramas. Os elementos são as abstracções que são sintetizadas em primeiro lugar no modelo; as relações são as formas de ligar os elementos; os diagramas agrupam colecções de componentes relacionadas.

A base dos modelos em UML são os diagramas. É com base neles que se identificam os elementos e as relações, já que estas dependem do tipo de diagrama que se está a criar. Em seguida são apresentadas os elementos, depois são descritos os tipos de relações e depois são descritos os vários tipos de diagramas.

### Elementos

Os elementos são as principais componentes UML. São usadas para escrever modelos bem formados. Existem quatro tipos de elementos:

- **Estruturais.** Os elementos estruturais são os nomes usados nos modelos UML. São as partes mais estáticas de um modelo, que representam elementos conceptuais ou físicos. Ao todo existem sete tipos de componentes estruturais (por exemplo, classes).

- **Comportamentais.** Os elementos comportamentais são as partes dinâmicas do modelo UML. São os verbos do modelo, que representam o comportamento no espaço e no tempo. Existem dois tipos primários de elementos comportamentais: mensagens e estados.

- **Associativos.** Os elementos associativos permitem organizar um conjunto de modelos UML. Só existe um tipo de elementos associativos que são os pacotes (*packages*).

- **Documentais.** Os elementos documentais são as partes explicativas dos modelos UML. São os comentários utilizados para descrever, ilustrar e remarcar qualquer elemento de um modelo. Apenas existe um elemento deste tipo, a nota.

### Relações

As relações são ligações entre elementos com características específicas. Existem cinco tipos de relações:

- **Dependências**. Relação semântica entre dois elementos, em que a alteração de um deles, pode afectar o outro.
- **Associações**. Relação estrutural que descreve um conjunto de ligações entre objectos.
- **Generalizações/Especializações**. Relação em que os objectos do elemento especializado são substituídos pelos objectos do elemento generalizado.
- **Realizações**. Relação semântica entre classificadores, em que um classificador especifica um contrato e outro garante que o cumpre.
- **Agregações/Composições**. É uma simples relação, em que existe uma distinção entre uma "parte"e o "todo". Uma composição tem o mesmo significado, só que obriga à existência da relação para que a "parte"possa existir.

### Diagramas

Os diagramas UML são a representação gráfica de um conjunto de elementos interligadas através relações segundo regras definidas pelo tipo de diagrama. Existem nove tipos de diagramas UML, que são apresentados na tabela 2.1. Contudo, apenas o primeiro tipo, o diagrama de classes, é que vai apresentado em detalhe, porque é o único que é usado neste projecto de tese. Os diferentes tipos de diagramas permitem visualizar o sistema de diferentes perspectivas, podendo-se dizer que os diagramas são uma projecção do sistema. Os diagramas restringem as combinações de elementos e relações. No entanto, as ferramentas de edição de diagramas UML não são rigorosas a esse ponto, permitindo em muitos casos fazer combinações que são incorrectas.

Os diagramas de classes são o tipo de diagramas mais comum na modulação de sistemas orientados a objectos. Um diagrama de classes representa classes, interfaces, colaborações e as suas relações.

Os diagramas de classes são usados para modular a visão estática de um sistema. Este processo envolve a identificação das componentes do sistema, bem como a caracterização das relações existente entre eles. Os diagramas de classes são também um ponto de partida para o desenho de outros tipos de diagramas, como por exemplo um diagrama de objectos, em que são representadas instâncias das classes.

São diagramas importantes, não só, para visualizar, especificar e documentar aspectos estruturais, mas também no processo de *forward* e *reverse engineering*. A partir deste tipo de diagramas é possível gerar documentos XMI com a respectiva informação, que podem ser usados por outras ferramentas noutras fazes do processo de desenvolvimento. É também possível desenhar diagramas de classes referentes a sistemas já implementados. Por exemplo, no desenho de uma aplicação Java, existe um mapeamento directo entre as classes Java e as classes UML.

As principais componentes, dos diagramas de classes são:

| Diagrama | Representa |
|---|---|
| **Classes** | Classes, interfaces e colaborações e as suas relações. |
| **Objectos** | Objectos e as suas relações. |
| **Utilização** | Utilizações por parte dos actores. |
| **Sequenciação** | Mensagens entre objectos, enfatizando o tempo e a ordem. |
| **Colaboração** | Objectos que enviam e recebem mensagens entre eles, enfatizando a estrutura. |
| **Estado** | Estados, transacções, eventos e actividades. |
| **Actividades** | Fluxo de actividade. |
| **Componentes** | Dependências relativas a um conjunto de componentes. |
| **Colocação** | Configuração dos nós do sistema. |

Tabela 2.1: Tipos de diagrama UML.

- **Classes.** Descrição de um conjunto de objectos que partilham os mesmos atributos, operações, relacionamentos e semântica.

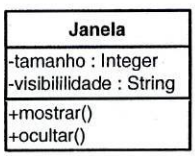



Figura 2.2: Classe UML.

- **Interfaces.** Conjunto de operações que especificam um serviço, uma classe ou uma componente. Descreve o comportamento visível do elemento. Pode representar todo o comportamento ou apenas uma parte. Define um conjunto de especificações de operações, mas nunca a implementação de operações.

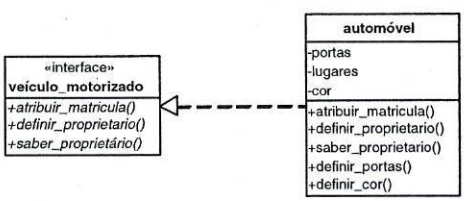



Figura 2.3: Interface UML.

- **Colaborações.** Define uma interacção entre entidades que cooperam para um determinado fim. É um conjunto de entidades em que o todo é maior do que a soma dos elementos.
- **Relações**
  - **Associações.** É uma relação estrutural que descreve um conjunto de ligações entre objectos.

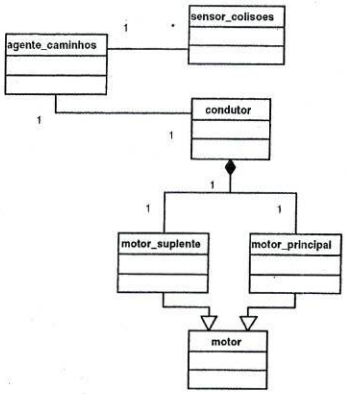



Figura 2.4: Colaboração UML.

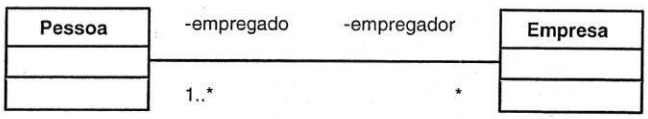



Figura 2.5: Associação UML.

- **Dependências.** É uma relação semântica entre dois elementos em que a alteração de um pode afectar a semântica do outro.

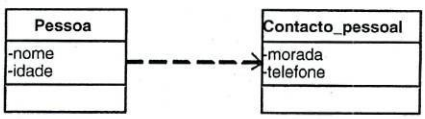



Figura 2.6: Dependência UML.

- **Generalizações.** É uma relação do tipo especialização/generalização, em que os objectos do elemento especializado são substitutos dos objectos do elemento generalizado. Desta forma, um elemento filho partilha a estrutura e comportamento do elemento pai.

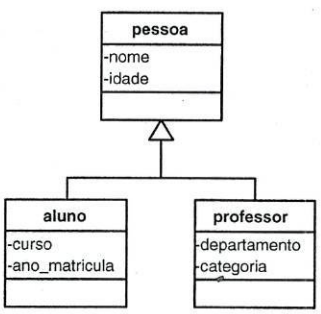



Figura 2.7: Generalização UML.

- **Realizações.** É uma relação semântica entre classificadores, em que um classificador especifica um contracto que outro classificador garante o seu cumprimento.

Como todos os diagramas UML, os diagramas de classes podem conter notas e restrições. Estes diagramas podem ainda conter pacotes, que são usados para agrupar elementos do modelo em diversas partes.

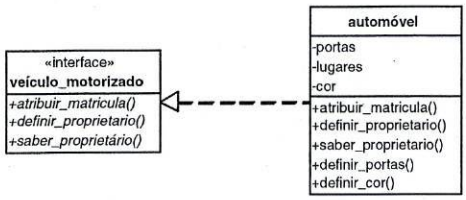



Figura 2.8: Realização UML.

### Mecanismos de especialização de diagramas

As classes são as componentes mais importantes dos sistemas orientados a objectos. Fazem parte de um conjunto de componentes UML (classe, interfaces, pacotes, etc) que são designados por classificadores. Dispõem de formas que permitem caracterizar melhor as suas componentes. A sua utilização permite definir a semântica dos classificadores até ao nível que se pretender. Entre essas formas de caracterização estão:

- **Multiplicidade.** Por principio uma classe tem um número indeterminado de instâncias. A multiplicidade serve para restringir esse número.

  A multiplicidade também se aplica aos atributos. Serve para restringir as instâncias do atributo e também para restringir o tipo de dados, mais propriamente, o tamanho do atributo.

- **Visibilidade.** A visibilidade é um dos principais mecanismos de especialização dos atributos e operações. A visibilidade de uma componente, de um classificador, especifica quando é que ela é visível por outros classificadores. Em UML, é possível especificar um destes três níveis de visibilidade:

  - **public.** A componente é visível por qualquer classificador; é especificada com o símbolo +.
  - **protected.** A componente é visível por qualquer descendente do classificador; é especificada com o símbolo #.
  - **private.** A componente é apenas visível pelo próprio classificador; é especificada com o símbolo -.

### Regras

As componentes UML, não podem simplesmente ligar-se umas com as outras, de qualquer forma. O UML caracteriza um conjunto de regras que especificam como é que um modelo bem formado deve ser. Um modelo bem formado é um modelo semanticamente consistente.

As regras semânticas são aplicadas sobre:

- **Nome.** Como identificar os elementos, os relacionamentos e os diagramas.
- **Competência.** O contexto que dá um significado específico aos nomes.
- **Visibilidade.** Como é que os nomes podem ser vistos e usados por outros.

- **Integridade**. Como é que os elementos se relacionam de forma consistente e própria.
- **Execução**. O que significa executar ou simular um modelo dinâmico.

#### Mecanismos de consistência

Como na construção de qualquer coisa, o UML dispõem de mecanismos de consistência, para garantir a boa formação dos modelos. Existem quatro tipos de mecanismos: Especificações, Adereços, Divisões Comuns e Mecanismos Extensivos.

O UML não é apenas uma linguagem gráfica. Por detrás de cada notação gráfica existe uma especificação textual da sintaxe e semântica da componente. Por exemplo, por detrás da representação de uma classe está a especificação dos atributos, operações, cardinalidades, estereótipos, tipos de dados, restrições, etc, enquanto que a sua representação apenas apresenta um sub-conjunto desta informação. A representação gráfica é usada para visualizar o sistema e a especificação é usada para definir os detalhes do sistema.

A especificação de uma classe pode incluir outros detalhes, como se é abstracta ou não. Esses detalhes podem ser representados como adereços gráficos ou textuais segundo a representação básica de uma classe. Cada elemento na notação UML começa como um símbolo básico, ao qual podem ser adicionados uma variedade de adereços específicos de cada símbolo.

Quase todas as componentes básicas em UML têm este tipo de dicotomia (classe/objecto). Uma classe é uma abstracção, enquanto que um objecto é uma manifestação concreta dessa abstracção. Tem de existir uma divisão entre o que é abstracto e o que é concreto.

Não é possível que uma linguagem fechada seja suficientemente expressiva para representar com um grau de detalhe arbitrário todos os modelos, em todos os domínios. Por este motivo o UML é aberto, permitindo extender a linguagem de forma controlada. Os mecanismos de extensão são os seguintes:

- **Estereótipos**. Novos blocos básicos derivados de outros já existentes, específicos ao problema.
- **Valores Associados**. Permite adicionar informação à especificação de um elemento.
- **Restrições**. Extende a semântica das componentes permitindo adicionar ou alterar restrições da especificação.

## 2.2 XML

Em 1994, Tim Berners-Lee fundou uma organização - chamada *World Wide Web Consortium (W3C)* - dedicada a desenvolver tecnologias para a *World Wide Web* não proprietárias. Um dos principais objectivos do W3C foi tornar a Web universalmente acessível[Dea02].

O W3C é também uma organização de estandardização. As tecnologias estandardizadas pelo W3C são designadas por Recomendações. Entre elas constam o HTML (*Hipertext Markup Language*), o CSS (*Cascading Style Sheets*) e o XML (*Extensible Markup Language*). As Recomendações não são propriamente produtos de software, mas sim documentos que especificam o papel, a sintaxe, as

regras, etc. de tecnologias. Antes de se tornar uma Recomendação W3C, um documento deste tipo passa por três fases: *Working Draft*, *Candidate Recommendation* e *Proposed Recommendation.*

### 2.2.1 Linguagens de anotação

O conceito notação é o método de inclusão de informação que caracteriza conjuntos de informação. Em 1969, uma equipa de investigadores apresentou a primeira linguagem de anotação moderna, o GML *Generalized Markup Language*, existindo algumas mais antigas, utilizadas no inicio dos anos 60 como o Tex e o Nroff. O GML era uma linguagem com o intuito de ser uma meta-linguagem - linguagem que pode ser usada para descrever outras linguagens. O GML já usava uma forma de definição dos documentos, os DTD's (*Document Type Definition*). Mais tarde, o GML tornou-se SGML (*Standard Generalized Markup Language*)[Dea02].

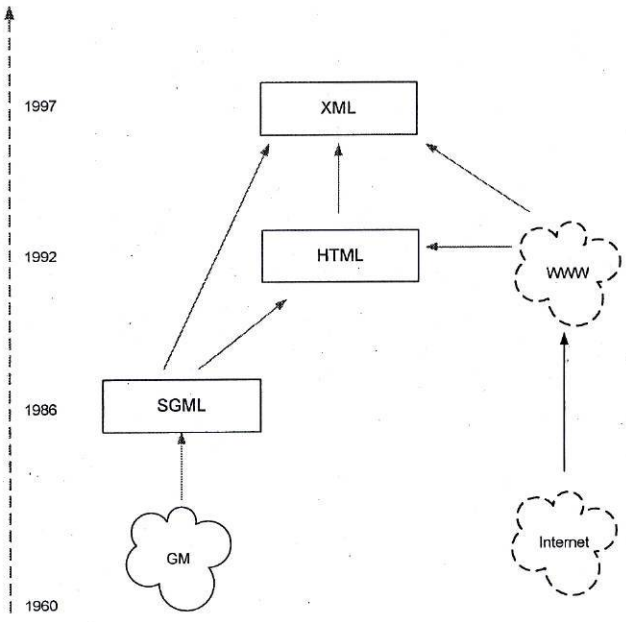



Figura 2.9: Evolução das linguagens de anotação.

O SGML é uma meta-linguagem que ficou bem estabelecida em 1986. Devido à sua grande robustez, manteve-se inalterada durante dez anos. Após este período, apenas pequenas alterações foram efectuadas à sua especificação. Contudo, a sua especificação era demasiado complexa, tornando o seu uso uma tarefa bastante complicada. Aplicações baseadas em SGML demoravam a sair para o mercado, continham diversos erros e muitas vezes estavam inacabadas, comparando com alguns sistemas proprietários. Muitos especialistas na área consideravam que era necessária uma versão do SGML simplificada, para tornar a abordagem às linguagens de anotação mais atractiva.

Nos últimos anos, a Internet tornou-se num dos principais meios electrónicos de consulta e troca de informação. Para que este meio proporcionasse uma forma de interligar e formatar documentos, surgiu a necessidade da existência de uma linguagem de anotação. O HTML (*Hypertext Markup Language*) é a linguagem que veio responder a essas necessidades. Esta linguagem foi criada a partir da sintaxe SGML, embora com princípios diferentes. Enquanto que o SGML é uma meta-linguagem, o HTML é uma linguagem de formatação e apresentação de documentos.

Com o passar do tempo, foi surgindo a necessidade de aumentar a especificação do HTML, para

abranger novas áreas de utilização. Foram adicionadas algumas extensões à sua especificação e criadas tecnologias, como o CSS para separar questões de forma e conteúdo das de estilo. Estas não eram o suficiente para atingir os novos objectivos, que por sua vez, iriam desviar o HTML do seu propósito inicial (formatação de documentos). Foi então criado o XML (*Extensible Markup Language*) pelo W3C. O XML é, tal como o SGML, uma meta-linguagem, que permite definir novas linguagens para aplicações particulares ou definir a organização de conjuntos de informação, separando de forma clara o conteúdo da apresentação da informação. As principais características do XML são apresentadas na tabela 2.2.

| Característica | Descrição |
|---|---|
| **Extensibilidade** | Conjunto de etiquetas extensível |
| **Apresentação vs. Conteúdo** | Etiquetas que descrevem conteúdos |
| **Vistas** | Múltiplas visões de um documento (proporcionado pelo XSL) |
| **Documento/Informação** | Suporte para documentos com uma infra-estrutura extensível para troca e validação de informação estruturada |
| ***Search/Query*** | Pesquisa através de *queries* baseadas em campos e alterações posteriores |

Tabela 2.2: Principais características do XML

A versão 1.0 do XML foi definida em 1998 como primeiro *Request for Comments* (RFC). Alguns erros, com significado irrelevante na documentação e algumas alterações em *standards* subjacentes, originaram a publicação da segunda versão em Outubro de 2000.

### 2.2.2 Núcleo do XML

A criação do XML, por parte do W3C, baseou-se em dez princípios que caracterizam a linguagem [Bea01]:

1. Possibilidade de ser usado sobre a Internet.
2. Suportado por um grande número de aplicações.
3. Compatível com o SGML.
4. Facilidade na construção de programas que processem XML.
5. O número de especificações adicionais deve ser mínimo.
6. Documentos XML devem ser humanamente legíveis e razoavelmente claros.
7. O desenho do XML deve ser preparado rapidamente.
8. O desenho do XML deve ser fácil e conciso.
9. Documentos XML devem ser fáceis de criar.

10. Dar pouca importância ao estilo das anotações XML.

O modelo abstracto de dados de um documento XML é uma árvore. Esta é definida por um conjunto de nós ligados hierarquicamente. Neste contexto, um nó pode ser uma qualquer componente da sintaxe do XML: elemento normal, atributo, entidade, texto, *namespace*, instrução de processamento e comentário. Existe apenas um nó de raiz, que é também designado por raiz do documento ou elemento de topo. A única relação existente entre nós é a de pai-filho, ou seja, um nó tem um único pai e um pai pode ter vários filhos. A figura 2.10 apresenta a árvore de nós do documento XML 2.1.

```
1  <?xml version = "1.0" ?> <pessoa>
2     <nome>
3        <primeiro_nome>Manuel</primeiro_nome>
4        <ultimo_nome>Silva</ultimo_nome>
5     </nome>
6     <morada>
7        <rua>Fernão de Magalhães</rua>
8        <localidade>Porto</localidade>
9     </morada>
10 </pessoa>
```

Programa 2.1: Exemplo de um documento XML.

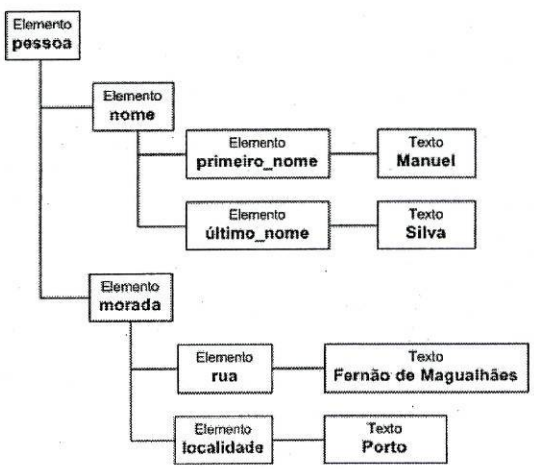



Figura 2.10: Árvore de nós do documento 2.1.

### Documento bem formado

Todos os documentos que respeitam a sintaxe 1.0 do XML são designados bem formados. Estes, podem ser usados sem que seja necessário um DTD (*Document Type Definition*) ou *XML Schema* que descreva a sua estrutura. Não podem conter referências externas ao documento, impossibilitando um tratamento especial dos atributos e a definição de valores por defeito.

Um documento bem formado contem um ou mais elementos que encaixam correctamente uns nos outros. Os elementos são representados por uma etiqueta inicial e outra final, podendo ser representados por apenas uma etiqueta com ambos os papeis. A boa formação de um documento tem de respeitar o modelo abstracto de dados. O conteúdo de um documento bem formado pode ser constituído por simples texto ou anotações.

A existência de documentos bem formados permite usar informação XML sem a necessidade de

construir e referenciar descrições externas da informação. O termo "bem formado"tem o mesmo significado que na lógica matemática - uma proposição é bem formada se cumpre as regras gramaticais, independentemente do valor lógico da mesma.

### Documento válido

Um documento XML é considerado válido se é bem formado, obedece às restrições de integridade definidas e respeita a gramática que descreve o seu conteúdo. Essa descrição pode ser feita de várias formas: através de DTD's, *XML Schemas*, etc.

Os DTD's usam uma gramática formal para descrever a estrutura e a sintaxe de documento XML. A regras de validação dos DTD's, designadas por restrições de integridade, garantem o bom relacionamento entre elementos, asseguram que os atributos têm valores válidos e que as referências a entidades estão correctamente definidas.

Desde o seu aparecimento, o XML tem vindo a ser usado em documentos complexos sobre a Web. Uma da áreas em que o XML começou a ser usado é a do comércio electrónico. Utilizações do XML, como estas, requerem uma forma mais robusta e complexa de organizar a informação, bem como permitir manter os tipos de dados quando se transfere informação de e para bases de dados relacionais. Estas necessidades tornam o uso de DTD's inadequado. Foi então criada uma forma de validação mais robusta, os *XML Schemas*[Wil01].

Os *Schemas* não vieram substituir os DTD's. Apenas introduziram uma nova forma de validação de documentos XML. Considerando as suas principais características, pode dizer-se que os DTD's deixam de ser usados. Por um lado, tudo o que os DTD's fazem, é possível fazer com os *Schemas*, por outro lado, a simplicidade dos DTD's pode originar grandes vantagens de utilização (por exemplo, a rapidez de definição de um DTD). A utilização dos *Schemas* deve acontecer quando é necessária uma definição dos dados mais robusta, extensível e com suporte de tipos de dados. A código 2.2 apresenta um exemplo de um documento XML, o código 2.3 apresenta definição DTD do documento anterior e o código 2.4 apresenta a definição do mesmo documento em *XML Schema*.

```
1  <?xml version = "1.0" ?>
2  <livro nome = "História do XML">
3     <capitulo nome = "Introdução">
4        <seccao nome = "Antes do XML">
5           As tecnologias...
6        </seccao>
7        <seccao nome = "Criação do XML">
8           ...
9        </seccao>
10       ...
11    </capitulo>
12
13    <capitulo nome = "Sintaxe do XML">
14       ...
15    </capitulo>
16    ...
17 </livro>
```

Programa 2.2: Documento XML.

```
1  <!ELEMENT livro ( capitulo+ )>
2
3  <!ELEMENT capitulo ( seccao+ )>
4  <!ATTLIST capitulo nome CDATA
```

```
5  #REQUIRED>
6
7  <!ELEMENT seccao ( #PCDATA )>
8  <!ATTLIST seccao nome CDATA #IMPLIED>
```

Programa 2.3: DTD do documento XML 2.2

```
1  <?xml version = "1.0" ?>
2   <schema>
3     <element name = "livro">
4        <complexType>
5           <attribute name = "nome">
6           <sequence>
7              <element ref = "capitulo"
8               minOccurs = "1"
9               maxOccurs = "unbounded"/>
10          </sequence>
11       </complexType>
12    </element>
13    <element name = "capitulo">
14       <complexType>
15          <attribute name = "nome"/>
16          <sequence>
17             <element ref = "seccao"
18              minOccurs = "1"
19              maxOccurs = "unbounded"/>
20          </sequence>
21       </complexType>
22    </element>
23    <element name = "seccao">
24       <complexType>
25          <attribute name = "nome"/>
26       </complexType>
27    </element>
28 </schema>
```

Programa 2.4: *Schema* do documento XML 2.2.

Uma das grandes diferenças entre os DTD's e os *Schemas* é que os DTD's são uma outra linguagem, enquanto que os *Schemas* usam a sintaxe do XML 1.0. Os DTD's são o principal conceito que o XML herdou do SGML.

### Analisadores sintáctico

Os analisadores sintácticos XML são as ferramentas utilizadas para fazer o seguinte:

- Verificar se um documento XML está bem formado.
- Verificar se um documento XML é válido.
- Aplicar uma transformação a um documento XML, originando outro.

O objectivo da criação de analisadores sitácticos XML foi o de retirar o processamento de documentos XML das aplicações, retirando-lhes algum peso. Os analisadores sintácticos devem apresentar os erros, ocorridos no processamento de documentos XML, às aplicações. Os erros podem dever-se à má formação dos documentos, ou à não validade dos mesmos. Para tornar os analisadores sintácticos XML mais simples, apenas são considerados estes dois tipos de erros.

### 2.2.3 Convenções

#### Namespaces

Um documento XML pode reunir elementos e atributos de diferentes definições, isto é, pode ser considerado a reunião de vários documentos XML. Como exemplo, pode-se considerar um documento SOAP, que contem um elemento chamado `Body`, que por sua vez pode conter elementos de outras definições XML. Por exemplo, o conteúdo desse elemento poderia ser XHTML[1]. Desta forma, um documento XML, reune pelo menos duas definições XML.

Considerando um documento com as características referidas, isto é um documento SOAP com um subconjunto XHTML, não existe uma diferenciação clara entre os elementos que pertencem a cada uma das definições, como demonstra o exemplo 2.5. Estas definições, individualmente, caracterizam um vocabulário de anotações. Vocabulários estes que podem coincidir em algumas anotações, originando colisões de nomes e problemas de reconhecimento dos mesmos. Tem de existir uma forma de distinção entre elementos de definições diferentes, para que as aplicações que processam estes documentos sejam capazes de verificar os elementos segundo a sua definição. O código 2.5 representa um exemplo de um documento que incorpora diferentes vocabulários.

```
1     <Body>
2       <getInfo>
3         <html>
4           <head>
5             <title>Test</title>
6           </head>
7           <body>
8             <p>This is only a test</p>
9           </body>
10        </html>
11      </getInfo>
12    </Body>
13  </Envelope>
```

Programa 2.5: Documento com anotações de diferentes vocabulários.

A solução para resolver este problema é associar um prefixo a cada definição, que vai ser usado nas respectivas anotações. Desta forma fica resolvido o problema do reconhecimento de anotações e o problema da colisão de nomes. Mas, esta solução tem ainda um problema, que é o da gestão dos prefixos relativamente às definições. Um prefixo tem de estar associado a apenas uma definição, mas uma definição pode ter associados vários prefixos. Isto obriga à existência de uma forma de identificar univocamente cada definição, que ficará associada ao prefixo.

Estes objectivos foram atingidos com a apresentação dos *Namespaces*, por parte do W3C. Como o próprio nome indica, os *Namespaces*, definem um espaço de nomes (vocabulário) com o mesmo contexto, ou seja, definem um espaço de nomes de anotações de uma definição. Os *Namespaces* são então definidos por um prefixo e um identificador único de uma definição. O W3C decidiu usar, para os identificadores, uma estrutura de nomes baseada no DNS (*Domain Name System*) da Internet, URI's *Uniform Resource Identifiers*).

Existem algumas dificuldades em assimilar a noção de *Namespace*. Muitas dúvidas e confusões surgiram à volta da real caracterização dos *Namespaces*, por isso talvez seja importante identificar alguns pontos que não caracterizam os *Namespaces*[Bea01]:

[1] O SOAP e o XHTML são tecnologias XML que são apresentadas em secções posteriores.

- Não existem fisicamente.
- Não é uma componente da sintaxe do XML 1.0.
- Não está directamente relacionado com DTD's ou *XML Schemas*.
- Não é uma redefinição de validação de informação XML.
- Não tem um significado semântico.
- Não é um objecto, um interface, ou qualquer coisa com uma estrutura interna específica.

Um *Namespace* deve ser definido como um atributo do elemento raiz do documento. O nome do atributo é `xmlns` seguido do caracter dois pontos (:) e do prefixo associado ao URI. O URI é o valor do atributo. É também possível definir um espaço de nomes por defeito. Neste caso não é necessária a existência de um prefixo, como se pode ver no exemplo no documento 2.6. Neste exemplo o elemento `inscrição` pertence à definição por defeito, enquanto que os elementos relativos à caracterização da pessoa, pertencem à definição com o prefixo `Pes`.

```
<documento xmlns:Pes = "http://www.teste.com/Pessoas"
           xmlns = "http://www.teste.com">
  <inscricao id = "I123">
    <Pes:nome>
      <Pes:primeiro_nome>António</Pes:primeiro_nome>
      <Pes:ultimo_nome>Silva</Pes:ultimo_nome>
    </Pes:nome>
  </inscricao>
</documento>
```

Programa 2.6: Utilização de *namespaces*.

Em seguida é apresentada o exemplo inicial (SOAP com XHTML), no documento 2.7, para se ter uma visão de como seria o documento utilizando *namespaces*.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <m:getInfo xmlns:m="Some-URI">
        <x:html xmlns:x="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
          <x:head>
            <x:title>Test</x:title>
          </x:head>
          <x:body>
            <x:p>This is only a test</x:p>
          </x:body>
        </x:html>
      </m:getInfo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Programa 2.7: Documento XML usando SOAP e XHTML com *namespaces*.

### XPATH

O XPath foi criado pelo W3C com o objectivo de definir um caminho sobre as componentes de um documento XML. Este caminho permite que outras tecnologias possam posicionar-se em determinados nós de uma árvore de um documento XML. O XPath dispõem de um conjunto de funções

de manipulação de *strings*, números, booleanos e conjuntos de informação, para atingir aos seus objectivos.

Na prática, o principal uso do XPath é como parte do XSLT (*Extensible Stylesheet Language Transformation*), nomeadamente na transformação de XML para HTML, ou na transformação de uma forma XML para outra qualquer. O XPath é usado sobre o XSLT na forma de expressões, que podem incorporar funções.

O XPath opera sobre a estrutura abstracta (árvore) do documento XML, em vez da sintaxe. Partindo de um ficheiro com um documento XML, é necessário criar uma representação abstracta do documento para que se possa usar o XPath. Esta representação é normalmente criada pelos *parsers* XML. As expressões do XPath baseiam-se nos nós da árvore do documento, em que a sua avaliação retorna o seguinte: conjuntos de nós, números, booleanos e textos. A avaliação das expressões não retorna nada, quando não existe um encaixe da expressão na representação abstracta do documento.

O XPath funciona da seguinte forma:

- Uma expressão ou função tem de ser aplicada a partir de um ou mais nós, que são designados por **nós de contexto**.
- Existe um **eixo** que define a direcção que é tomada sobre a estrutura hierárquica, a partir de cada nó de contexto.
- O **nó de teste** refina ou filtra o conjunto de nós, excluindo nós que não passem no teste.
- No caso de existirem *predicados*, estes refinam o conjunto de nós depois obtidos após as fases anteriores.
- Depois do primeiro passo de localização ter concluído, a informação relativa ao conjunto de nós obtido é retornado à aplicação, a não ser que existam mais nós de contexto para recomeçar a percorrer a arvore.

Como exemplo, é apresentada uma expressão XPath, no código 2.8, como parte de um elemento XSL (o conteúdo do atributo `select`). O objectivo é saber o primeiro nome da pessoa cuja inscrição é a "I123".

```
1 <xsl:value-of
2 select="/documento/inscricao[@id='I123']/nome/primeiro_nome" />
```

Programa 2.8: Uso de uma expressão XPath.

### 2.2.4 Meta-Linguagens

Da mesma forma que o XML veio responder a algumas necessidades, outras começaram a aparecer com a sua utilização. Tirando partido de uma linguagem robusta, criou-se um conjunto de novos *standards* que usam a especificação XML, para responder às necessidades prioritárias. Desta forma conseguiu-se tirar partido de uma linguagem bem definida e das ferramentas de edição de XML. Destas especificações vão ser apresentadas: o XSL, o XSLT e *XML Schema*. O XQuery é outro *standard* que vai ser apresentado, mas este não usa a sintaxe XML.

### Transformação de documentos

Uma das grandes vantagens da utilização do XML é o facto de ele ser um *standard*, que permite representar qualquer tipo de informação de uma forma genérica. Muitas das vezes essa representação de informação não é a mais conveniente, por isso, surgiu a necessidade de converter a informação para formatos de representação como o HTML ou o PDF.

Antecipando esta necessiade, a comunidade XML respondeu a este problema com a criação de uma linguagem XML designada por *Extensible Stylesheet Language* (XSL). O XSL incorpora duas especializações o *XSL-Formatting Objects* (XSF-FO) e o *XSL-Transformation* (XSLT).

O XSL-FO é uma linguagem de formatação, dirigida à apresentação visual do XML. O XSLT é uma linguagem de especificação de transformações de documentos XML. Permite aplicar um conjunto de regras de transformação sobre um documento XML, gerando um novo documento. O funcionamento básico do XSLT é a substituição de porções XML do documento fonte pelo código definido nas regras. Os documentos XSLT são também designados por folhas de estilo.

As regras, definidas num documento designado por folha de estilos, são aplicadas sobre um documento XML gerando um novo documento. O processo de aplicar as regras sobre o documento fonte é designado por transformação. O uso de regras de transformação é totalmente diferente do uso de procedimentos. Enquanto que o XSLT é uma linguagem declarativa, são declaradas as intenções através de regras, o uso de procedimentos obriga à reescrita de código sempre que se altere a estrutura do documento fonte ou o seu processamento.

Para se perceber bem o funcionamento do XSLT, é necessário conhecer bem a noção de transformação. Uma transformação não modifica o documento fonte. Into é, existe um documento sobre o qual se aplica uma transformação que tem como resultado um documento novo. Uma transformação é composta por um conjunto de regras. Não é um processo algorítmico das tradicionais linguagens de procedimentos. É um conceito idêntico ao das transformações matemáticas. Não existe um processamento passo-a-passo. Existe sim a aplicação de um conjunto de regras.

Uma regra é composta por um padrão de encaixe no documento fonte e por um ou mais elementos de geração de resultados. Quando o processador encontra um padrão de encaixe no documento fonte, gera um resultado a partir dos elementos da regra. Para definir os padrões de encaixe é usado o XPath. Como foi apresentado anteriormente, o XPath especifica uma sintaxe para definir caminhos sobre um documento XML. Tais caminhos permitem alcançar os nós pretendidos num documento.

O documento 2.9 apresenta uma transformação, do documento XML 2.10, para HTML. O resultado dessa transformação é apresentado no documento 2.10, com a visão do documento 2.11.

```
1  <?xml version = "1.0" ?>
2
3  <xsl:stylesheet version = "1.0"
4     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
5
6     <xsl:template match = "/">
7        <html>
8           <xsl:apply-templates/>
9        </html>
10    </xsl:template>
11
12    <xsl:template mach = "pessoa">
13       <head>
14          <title>Pagina do <xsl:value-of select = "nome/primeiro_nome">
```

```
15                          <xsl:value-of select = "nome/ultimo_nome">
16        </head>
17        <body>
18           <h1>Dados Pessoais<h2>
19           <br>
20           <table border = "1">
21              <tr>
22                 <th>Nome</th>
23                 <td><xsl:value-of select = "nome/primeiro_nome"></td>
24              </tr>
25              <tr>
26                 <th>Apelido</th>
27                 <td><xsl:value-of select = "nome/ultimo_nome"></td>
28              </tr>
29              <tr>
30                 <td> </th>
31                 <td> </td>
32              </tr>
33              <tr>
34                 <td>MORADA</th>
35                 <td> </td>
36              </tr>
37              <tr>
38                 <th>Rua</th>
39                 <td><xsl:value-of select = "morada/rua"></td>
40              </tr>
41              <tr>
42                 <th>Localidade</th>
43                 <td><xsl:value-of select = "morada/localidade"></td>
44              </tr>
45           </table>
46        </body>
47     </xsl:template>
48  </xsl:stylesheet>
```

Programa 2.9: Transformação XSLT para HTML.

```
1  <html>
2     <head>
3        <title>Pagina do Manuel Silva</title>
4     </head>
5     <body>
6        <h1>Dados Pessoais<h2>
7        <br>
8        <table border = "1">
9           <tr>
10             <th>Nome</th>
11             <td>Manuel</td>
12          </tr>
13          <tr>
14             <th>Apelido</th>
15             <td>Silva</td>
16          </tr>
17          <tr>
18             <td> </th>
19             <td> </td>
20          </tr>
21          <tr>
22             <td>MORADA</th>
23             <td> </td>
24          </tr>
25          <tr>
26             <th>Rua</th>
27             <td>Fernão de Magalhães</td>
28          </tr>
29          <tr>
```

Dados Pessoais

| | |
|---|---|
| **Nome** | Manuel |
| **Apelido** | Silva |
| | |
| **MORADA** | |
| **Rua** | Fernão de Magalhães |
| **Localidade** | Porto |

Figura 2.11: Visão do documento 2.10.

```
30                <th>Localidade</th>
31                <td>Porto</td>
32              </tr>
33            </table>
34      </body>
35  </html>
```

Programa 2.10: Documento HTML resultante da transformação XSLT.

Para além das substituições definidas nas regras, o XSLT permite outros tipos de funcionalidades.

- É possível usar variáveis, mas apenas como nas linguagens funcionais, ou seja, só se pode definir um único valor de uma variável por cada ramo da árvore do documento.

```
1  <xsl:variable name="x" select="/Pessoa/nome@primeiro_nome">
```

Programa 2.11: Variáveis no XSLT.

- As regras podem ser compostas por elementos que permitem iterações, que operam sobre os nós, de acordo com o padrão de encaixe. Este tipo de iterações não incrementam variáveis como as linguagens de procedimentos.

```
1  <xsl:for-each select="/Pessoa/Nome">
2     ...
3  </xsl:for-each>
```

Programa 2.12: Elementos de iteração do XSLT.

- Também existem elementos que permitem fazer testes condicionais.

```
1  <xsl:if test="position()>1">
2     ...
3  </xsl:if>
```

Programa 2.13: Elemento para fazer testes condicionais no XSLT.

- Como qualquer especificação XML, a modularidade é sempre permitida, bastando para isso importar documentos com outras definições.

```
1  <xsl:import href="qualquercoisa.xsd" />
```

Programa 2.14: Importação de definições em XSLT.

- Podem-se definir procedimentos e usar recursão.

```
1  <xsl:template name="teste">
2    <xsl:param name="x">
3    <xsl:if test="number($x)>0">
4      <xsl:call-template name="test">
```

```
5         <xsl:with-param name="x" select="number($x) - 1" />
6       </xsl:call-template>
7     </xsl:if>
8   </xsl:template>
```

Programa 2.15: Procedimento em XSLT.

- Existem extensões que possibilitam o uso de outras linguagens (por exemplo, JAVA) nas regras, podendo assim ultrapassar as limitações do XSLT.

```
1  <xsl:stylesheet version="1.0"
2    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    xmlns:date="http://www.oracle.com/XSL/Transform/java/java.util.Date">
4      ...
5    <xsl:value-of select="date:toString(date:new())" />
6      ...
```

Programa 2.16: Extensões em XSLT.

- A possibilidade de gerar mais do que um documento como resultado, também é possível através do uso de extensões.

```
1  <xsl:result-document
2   href="{$directory_path}/{$package_name}.xsd" format="temp">
3     ...
4  </xsl:result-document>
```

Programa 2.17: Geração de vários documentos usando XSLT.

- Inclui elementos para geral XML directamente.

```
1  <xsl:element name="Pessoa">
2    <xsl:attribute name="nome"><xsl:value-of select="@primeiro_nome"/></xsl:attribute
       >
3    <xsl:value-of select="." />
4  </xsl:element>
```

Programa 2.18: Geração de XML em XSLT.

### Interrogação de dados

Com o passar dos anos o XML foi ganhando popularidade muito rapidamente. O aparecimento de *standards* baseados em XML veio contribuir para o aumento da sua utilização ao nível da troca de documentos e armazenamento de informação[dBW, Len, Bea01].

À medida que foi aumentando a utilização do XML, surgiu a necessidade de consultar informação em documentos XML de forma mais eficiente e efectiva. Para isso era necessário criar uma linguagem em que fosse possível especificar exactamente que informação se pretende obter ou alterar dos documentos XML.

Foi proposto um *standard* XML de interrogações, o XPath, para obter informações dos documento XML. Contudo, a linguagem não era suficientemente sofisticada para suportar consultas mais complexas, que os utilizadores estavam habituados a fazer com linguagens como o SQL nas bases de dados relacionais. Para além da capacidade de seleccionar informação de vários documentos era também necessária uma forma de definir o formato do resultado. Foi então que o W3C Query Working Group

desenvolveu o XQuery como resposta a estas necessidades, em que o seu estado actual é *Working Draft*, mas que em breve se tornará numa *W3C Remommendation.*

O XQuery é uma linguagem declarativa como o SQL e o XSLT. Ao contrário das linguagens de programação usuais como o Java e o C, o XQuery não segue uma estratégia de avaliação particular. Apenas define o que se pretende obter do documento fonte e não a forma de obter.

As *queries* têm de seguir o modelo de dados do XQuery. Isto quer dizer que o resultado de uma *query* pode sempre ser usada como *input* de uma nova *query*. Isto acontece porque os dados de entrada e de saída são ambos instâncias dos mesmos modelo de dados. As *queries* podem seleccionar partes de árvores, fazer transformações arbitrárias, construir novas instâncias e fazer operações que preservem a hierarquia e sequência do documento XML original. O XQuery permite fazer operações sobre vários documentos XML.

O modelo de dados do XQuery define a informação de entrada e de saída de um processador de XQuery. É uma forma abstracta de ver os documento XML; define que aspectos dos documento XML são significantes. Já forma referidos alguns modelos de dados, nomeadamente na navegação do XPath ou transformações do XSLT. De acordo com as suas necessidades, as aplicações XML, têm usado o modelo de dados que mais lhes convém, originando alguns conflitos nesta área.

O W3C está a tentar encontrar uma solução para resolver este problema. Ela passa por unificar os modelos de dados usados pelas várias especificações do W3C. O *Infoset* (*XML Information Set*) é a primeira tentativa para proporcionar um conjunto de definições que podem ser usadas noutras especificações que necessitam de referenciar a informação num documento XML bem formado. Em vez de proporcionar uma lista de todas as especificações possíveis que uma aplicação possa considerar importantes, o *Infoset* define um subconjunto comum a múltiplas especificações.

O modelo de dados do XQuery é baseado no *Infoset* com o suplemento dos tipos de dados dos *XML Schemas.* Uma instância do modelo de dados do XQuery pode ser constituída por nós, valores simples, ou sequências. Também existe a noção de colecção e referência. As colecções são usadas para modelar múltiplos documentos XML ou fragmentos XML. As referências são usadas pelos operadores para referir partes internas ou externas ao documento. Um exemplo das internas é o uso de atributos do tipo `ID`. Um exemplo das externas é o uso de referências *XLink*, para ligar partes de diferentes documentos.

Os nós podem ser dos tipos referidos na apresentação do XML. Os valores simples podem ser um dos dezanove tipos de dados primários definidos pelo *XML Schema.* Uma sequência pode ser considerada como uma lista de nós e valores. Uma sequência pode modelar uma instância do modelos de dados representando colecções de documentos, conjuntos de nós como no XPath, ou uma lista simples de valores. Outra característica importante das sequências é o facto do modelo de dados não destingir um nó (ou valor) de uma sequência que contém esse único nó.

O XQuery é uma linguagem funcional em que a *query* é representada por uma expressão. As expressões das *queries* podem estar encaixadas, da mesma forma do SQL. As expressões do XQuery aumentam as vantagens da utilização do XML, já que permitem especificar o formato dos dados de resultado.

As expressões do XQuery mais utilizadas são as seguinte:

- Expressões de caminhos

- Construção de elementos
- Expressões FLWR (*For/Let/Where/Return*)
- Expressões envolvendo operadores e funções
- Expressões condicionais
- Expressões quantificadas
- Expressões que testam ou modificam tipos de dados

Observando alguns exemplos da utilização do XQuery, nos documentos 2.19 e 2.20, as semelhanças com o XSLT são evidentes. De facto, muitas das vezes o XSLT desempenha o papel do XQuery, gerando alguma controvérsia quanto à sobreposição de funcionalidades entre ambas as linguagens. Actualmente o XSL e o XML *Query Working Group* estão a tentar solucionar este problema até certo ponto, com o objectivo de tornar o XPath 2.0 compatível com o XSLT e a XML Query Language. Isto vai obrigar à existência de cedências por parte de ambos, originando que as linguagens se tornem ainda mais similares.

```
1 document("biblioteca.xml")/livro/capitulo[@nome = "Introdução"]
```

Programa 2.19: Utilização do XQuery.

```
1 <seccoeslivros>
2   FOR $y IN (/livro/capitulo/*)
3   RETURN
4       <seccao>$y</seccao>
5 </seccoeslivros>
```

```
1 <seccoeslivros>
2   FOR $y IN (/livro/capitulo/*)
3   RETURN
4       <seccao>$y</seccao>
5 </seccoeslivros>
```

Programa 2.20: Outra hipótese de utilização do XQuery.

A especificação do XQuery não impõem nenhuma restrição relativamente à sintaxe. Foi criada uma sintaxe mais apelativa para as pessoas. Contudo, também existia a necessidade de uma sintaxe XML, para ser mais facilmente utilizada nas aplicações e para tirar partido das ferramentas XML existentes. O XQueryX é a sintaxe XML do XQuery. Esta nova sintaxe veio permitir que o as expressões sejam definas em XML e que o formato do resultado das *queries* também o sejam. Existem vantagens em usar esta sintaxe nas seguintes situações:

- **Gerar *Queries*.** Em ambientes de programação que usem XML, pode ser mais conveniente usar *queries* na representação XQueryX do que a representação original do XQuery, para tirar partido das ferramentas XML.
- ***Queries* dentro de *Queries*.** Como o formato das *queries* e do resultado é XML, podem ser feitas novas *queries* sobre os resultados.
- **Incorporar *queries* em XML.** Podem ser incorporadas *queries* XQueryX directamente num documento XML.

#### Definição de tipos de dados

O *XML Schema* é uma linguagem, criada pelo W3C, para descrever a estrutura e o conteúdo de documentos XML. Tem o mesmo propósito dos DTD's, mas dispõem de mais recursos para melhor caracterizar um documento XML. Embora os DTD's não deixem de existir, os *Schemas* conseguem abranger mais áreas de utilização[TGBS99, Bea01].

A definição dos *Schemas*, por parte da equipa do W3C, tinha como objectivos iniciais:

- Criar mecanismos de validação da estrutura de documentos XML, análoga à dos DTD's.
- Criar mecanismos de validação especificados segundo a sintaxe XML.
- Suportar um conjunto mais vasto de tipos de dados, relativamente aos DTD's.
- Permitir aos utilizadores definir novos tipos de dados.
- Integração de *namespaces*.
- Permitir herança de elementos, atributos e definições de tipos de dados.
- Distinguir a representação léxica da informação de um conjunto de informação subjacente.

Após a sua implementação, a questão mais relevante era comparar os *Schemas* com os DTD's e verificar as inovações apresentadas pelos primeiros. Ao nível de recursos, foram notórias as vantagens dos *Schemas* relativamente aos DTD's. Contudo a utilização dos *Schemas* é um pouco mais complexa, necessitando de algum tempo para cativar os utilizadores indecisos.

Os principais pontos que caracterizam este *standard* estão expressos na tabela 2.3.

As componentes do *Schema* correspondem aos blocos que compõem o modelo abstracto de dados do *Schema*. Ao todo existem doze componentes, divididas em três grupos:

- **Componentes Primárias**: Definição de tipos simples e complexos, e declaração de elementos e atributos.
- **Componentes Secundárias**: Grupos de atributos, restrições de identidade, definições de grupos de modelos e declaração de anotações.
- **Componentes de Ajuda**: Diferem das anteriores, pois, não podem ser caracterizadas ou independentemente acedidas. São as anotações, os grupos de modelos e as *wildcards*.

O grupo que é mais usado é o das componentes primárias. Existem quatro tipos de componentes neste grupo, como indica tabela 2.4.

### 2.2.5 Utilizações do XML

Quando o XML foi criado pelo W3C, existia uma lista de princípios que caracterizavam a linguagem. Contudo, a criação de uma nova linguagem tem de ter associado, à partida, um conjunto de aplicações

| Característica | Descrição |
|---|---|
| **Tipos de Dados** | Suporta uma grande variedade de tipos de dados. |
| **Sintaxe XML** | Os *Schemas* são escritos em XML, aproveitando uma linguagem existente e todas as ferramentas de trabalho, ao contrário dos DTD's que são outra linguagem. |
| **Modelos de Conteúdos** | Eficiente suporte para modelos de conteúdos, permitindo organizações de informação mais complexas. |
| **Extensibilidade** | Permite re-usar partes de *Schemas* noutros *Schemas*. Definir tipos complexos que podem ser usados em diferentes *Schemas*. Definir novos tipos de dados a partir de tipos existente. Referenciar vários *Schemas* a partir de um único documento. |
| **Documentação própria** | O *Schema* pode conter elementos de anotações para descrever o propósito das componentes, documentando assim o *Schema*. |

Tabela 2.3: Principais Características do *XML Schema*

para a mesma, de forma a justificar a sua criação. A extensibilidade que caracteriza o XML, leva à possibilidade de utilização da linguagem em diversas aplicações, mesmo que esta não seja a mais adequada, mas existem aplicações concretas que têm todas as vantagens em usar XML.

### Representação de documentos

O XML pode ser usado para representar documentos semi-estruturados, como guias de trabalho, manuais técnicos, catálogos, jornais, relatórios, etc. Para além de outras utilizações do XML na publicação de documentos ou representação de informação, existe uma aplicação muito actual e especializada, que é a nova geração de telefones móveis.

Todas as características de um documento típico podem ser representadas por etiquetas XML, de diversas formas. As seguintes especificações usam o XML para a representação de documento: o XHTML, o MathML, o SMIL, etc.

### Ficheiros de configuração

Muitas aplicações estão a adoptar a linguagem XML nos seus ficheiros, nomeadamente os de configuração. Estes ficheiros têm grandes vantagens em usarem XML, pois necessitam que a informação esteja estruturada, que seja simples alterar as configurações, que exista documentação dos ficheiros, que existam ferramentas eficazes para processar os ficheiros e que sejam independentes da plataforma.

Como exemplo de uma aplicação que usa XML nos ficheiros de configuração é o *Tomcat 4* do projecto

| Componente | Exemplo |
|---|---|
| **Declaração de elementos** | `<xsd:element name="nome"`<br>`  type="xsd:string"`<br>`  minOccurs="0"`<br>`  maxOccurs="1"/>` |
| **Definição de tipos simples** que apenas suportam valores, into é, não pode incluir elementos | `<xsd:simpleType name="nota">`<br>`  <xsd:restriction base="xsd:string">`<br>`  <xsd:maxLength value="2"/>`<br>`  </xsd:restriction>`<br>`</xsd:simpleType>` |
| **Definição de tipos complexos** que podem ter sub-elementos e atributos | `<xsd:complexType name="Disciplina">`<br>`  <xsd:all>`<br>`  <xsd:element name="cad"`<br>`  type="data:Disciplina_oid"/>`<br>`  <xsd:element name="nome"`<br>`  type="data:outro_texto"/>`<br>`  </xsd:all>`<br>`</xsd:complexType>` |
| **Declaração de atributos** | `<xsd:attribute name="data"`<br>`  type="xsd:date"`<br>`  use="required"/>` |

Tabela 2.4: Componentes principais dos *Schemas*

*Jakarta* da *Apache Software Foundation.* O *"conf/server.xml"* é o principal ficheiro de configuração do servidor.

### Transformação

A adopção do XML por parte das aplicações, pode resultar num benefício com alguns custos, devido ao alto nível de automação necessário para converter a informação do formato original para outros - em alguns casos são necessários processos automáticos para publicar a informação em formatos adequados, como o papel, CD-ROM ou Internet.

A vantagem do uso do XML, é o seu não isolamento. Para além da especificação do seu núcleo, o XML incluí *standards* opcionais que permitem transformar XML noutros formatos (o XSL e XSLT). Desta forma, a mesma informação pode ser representada de diferentes formas.

### Partilha e Armazenamento de Dados

Alguns observadores da indústria proclamaram o XML como a solução para resolver os problemas de partilha de informação. Diziam que o XML, juntamente com o XSL, permitiriam a completa interoperabilidade de conteúdos entre diferentes aplicações e plataformas. Na realidade, o XML veio apenas contribuir para solucionar alguns desafios que a partilha de informação apresenta[SR01]. Os principais desafios, da partilha de informação, e os contributos do XML para os resolver, são apresentados na tabela 2.5. Um exemplo de uma utilização de documentos XML para partilha de dados é o RDF (*Resource Description Framework*).

| Desafios | Contributos |
|---|---|
| **Distribuição geográfica** | Permite invocação remota de funções - por exemplo, via SOAP. |
| **Estruturas de dados e linguagens heterogéneas** | Proporciona uma sintaxe neutra permitindo ser usada em estruturas de dados heterogéneas. |
| **Representação de atributos e semânticas heterogéneas** | Permite adicionar meta-informação para descrever a semântica e representação de informação. |
| **Esquemas heterogéneos** | Ferramentas de simples utilização que encorajam a reutilização de recursos. Permite um mapeamento simples para um qualquer esquema bem definido. |
| **Identificação de Objectos** | Através do uso de meta-informação, consegue adicionar informação que permite identificar objectos distintos. |
| **Reconciliação de valores** | Com o uso de meta-informação, deixam de existir dúvidas quanto ao significado de alguns valores. |

Tabela 2.5: Contributo do XML nos desafios da partilha de informação.

O XML e as tecnologias de bases de dados complementam-se em vez de competirem. Enquanto que o XML assenta na representação de informação semi-estruturada, as tecnologias de bases de dados estão mais capacitadas para trabalhar com informação estruturada e aumentar a integridade da informação.

Na publicação de informação na Web, no comércio electrónico, ou noutras aplicações, os programadores podem criar diversas visões da informação em XML, que permitem uma avaliação no momento. Na troca de informação entre diferentes tecnologias de bases de dados, o XML pode ser usado como formato intermédio. A simples publicação da informação, das bases de dados, em XML pode facilitar a compreensão, já que a informação fica representada num formato genérico e familiar, contendo um esquema com a sua descrição.

Em conclusão pode-se dizer que as tecnologias estão a adoptar ideias do XML, para tornar uma relação mais próxima. A comunidade de desenvolvimento do W3C reconheceu esta sinergia e está a adaptar ideias para usar *XML Schemas* e tecnologias XML de pesquisa. Por outro lado têm

sido desenvolvidas algumas tecnologias de bases de dados com o objectivo único de trabalhar com informação XML.

### 2.2.6 Aplicações

A extensibilidade do XML permitiu criar linguagens de anotações de acordo com as necessidades. Foi assim que surgiram as linguagens baseadas em XML. Desta forma podem tirar partido das características do XML, em particular as linguagens com objectivos bem definidos. O XHTML, o SVG e o XMI foram algumas destas linguagens, que serão apresentadas em seguida.

#### Extensão do HTML

As páginas Web, como todos os outros conteúdos da Internet funcionam segundo um conjunto de *standards* desenvolvidos para que os programas de visualização dessas páginas (os *browsers*) possam interpreta-las correctamente. Nos primeiros anos de vida da Internet, esses *standards* tinham bastantes defeitos, dificultando o desenvolvimento dos *browsers*, já que estes tinham de ser capazes de entender HTML ambíguo e apresentar as páginas o melhor possível. Esses tempos já passaram porque os *standards* foram melhorando[oPS02].

O W3C produziu uma recomendação de uma aplicação de XML designada por *Extensible HyperText Markup Language* 1.0 (XHTML). A implementação obedece a todas as regras gramaticais do XML (elementos correctamente encaixados, uso de aspas nos atributos, entre outros), enquanto integram o vocabulário do HTML (os elementos e atributos disponíveis e as relações existentes entre eles e com a mesma semântica visual). O XML é uma forma de adicionar uma estrutura às páginas Web, permitindo o uso de descrições únicas das páginas Web. O XML é um *standard* rigoroso, inflexível - qualquer erro origina uma falha de documento. O rigor introduzido pelo XHTML tem alguns custos, já que os principais *browsers* têm de suportar esta nova linguagem, contudo existe a certeza que as páginas passam a ter a mesma representação em qualquer *browser*.

O XHTML já é suportado pelos *browsers* mais usados como o Internet Explorer, o Netscape e o Opera. Os pontos seguintes descrevem as diferenças existentes entre o tradicional HTML e o XHTML.

- Os documentos têm de ser bem formados, todos os elementos têm estar propriamente terminados, e os elementos têm de estar propriamente encaixados.
- O nome dos elementos ou atributos têm de estar em letras minúsculas.
- Elementos vazios têm de ter uma etiqueta de finalização ou então uma "/"no fim da etiqueta de inicio.
- Os valores dos atributos têm de estar sempre entre aspas.
- Não podem existir atributos sem valor.
- É possível usar espaços e mudanças de linha nos valores dos atributos.
- Os atributos podem ter listas de valores possíveis e valores por defeito.
- É possível usar caracteres representados como valores em hexadecimal.

- É possível identificar elementos proibidos. Pode-se retirar a possibilidade de ter determinados elementos contidos noutro.
- Os comentários dos elementos devem ser colocados em secções CDATA para evitar *parsings* incorrectos.
- O atributo `id` deve ser usado como identificador do elemento.

### Gráficos vectoriais

O SVG (*Scalable Vector Graphics*) é uma especificação XML que descreve gráficos de duas dimensões. O SVG suporta três tipos de objectos gráficos: formas gráficas vectoriais (por exemplo, caminhos compostos por linhas rectas e curvas), textos e imagens.

Qualquer imagem vista num ecrã, independentemente do formato ser vectorial ou *bitmap*, é vista como um vector de *pixels* coloridos de duas dimensões. A diferença entre gráficos *bitmap* e gráficos vectoriais não está no formato da representação final - que é sempre *bitmap* - mas sim na forma de armazenamento, de implementação e no tempo que demora a ser feita a exibição do objecto gráfico.

Os objectos gráficos SVG, armazenados no formato XML, descrevem a forma como o objecto é desenhado ou pintado. Uma ferramenta de visionamento de objectos gráficos SVG é apenas uma motor de interpretação, que traduz a informação vectorial de uma imagem, para uma representação *bitmap*, a ser apresentada no ecrã. Esta tradução pode ser feita por parte da máquina do cliente de forma dinâmica, possibilitando ajustar alguns parâmetros de interpretação de acordo com as necessidades do cliente.

A especificação SVG oferece um rico conjunto de funcionalidades para manipular objectos gráficos vectoriais. O desenho de objectos SVG pode ser interactivo e dinâmico. As animações podem ser definidas e accionadas de forma declarativa (juntando animações SVG a conteúdos SVG) ou via *scripting*. É possível implementar um grande conjunto de animações, recorrendo a linguagens de *scripting* adicionais (como o JavaScript) sobre o SVG *Document Object Model*, proporcionando um completo acesso a todos os elementos, atributos e propriedades. A qualquer objecto gráfico SVG pode estar associado um grande conjunto de eventos, e pelo facto do SVG ser compatível com outros *standards* Web, em principio pode ser usado *scripting* em XHTML e SVG ao mesmo tempo, na mesma página Web. É oferecido um enorme potencial.

Estas capacidades permitem a criação de um leque alargado de gráficos sofisticados, animados ou estáticos. O tamanho de um ficheiro SVG é normalmente pequeno, comparando com os outros tipos de ficheiros gráficos, necessitando de uma menor largura de banda.

### Meta-informação XML

XMI é o acrónimo de *XML Metadata Interchange*. Embora seja uma tecnologia da OMG (*Object Management Group*), é baseado no *standard* XML do W3C[Bro99, Kum99, XMI].

O XMI define um conjunto de regras, XML, de representação de informação, para aplicações orientadas por objectos, como: ferramentas de desenho (UML), ferramentas de desenvolvimento (Visual Age para Java), bases de dados (Oracle), aplicações (Java, C++), repositórios (Visual Age da IBM,

Unisys Universal Repository), geradores de relatórios e documentos (*web browsers*). A figura 2.12 representa a relação entre as aplicações que tiram partido do uso do XMI. A sua definição usa a especificação XML directamente, mas não a modifica ou extende. O XMI é um *standard* baseado no conjunto de recomendações XML disponibilizadas pelo W3C com uma submissão final a 20 de Outubro de 1998. O XMI foi desenhado para ser compatível com as novas tecnologias XML. Isto inclui *Namespaces*, *XML Schemas*, etc. Em particular, o XMI usa características dos *Schemas*, como os tipos de dados e os mecanismos melhorados para a criação de tipos de documento. O XMI adoptará essas novas características, à medida que forem sendo introduzidas no XML pelo W3C.

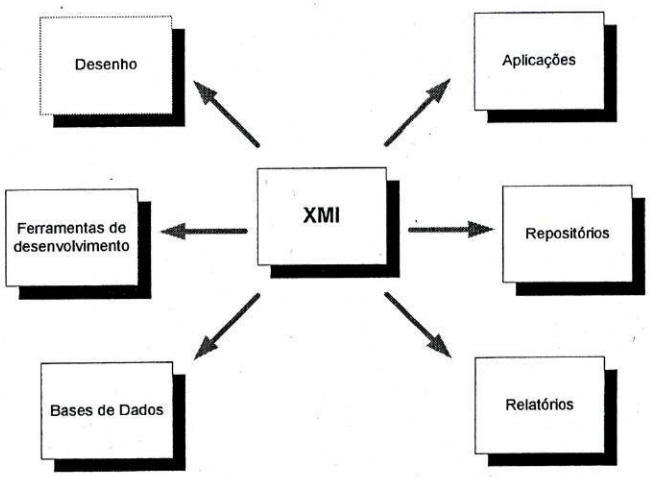



Figura 2.12: Posição do XMI relativamente a algumas aplicações.

Porquê uma linguagem de troca de informação baseada em XML?

O XML é uma linguagem simples, flexível em formato de anotações para troca de informação. O grande factor da sua escolha, em relação a outros formatos de dados, é a capacidade de separar informação de meta-informação e apresentação de conteúdo. Esta arquitectura de separação contextual permite que as ferramentas genéricas (como os *parsers* XML) validam um documento XML segundo a sua gramática (DTD, *Schema*). Outro das razões para a escolha de uma linguagem baseada em XML é o facto de se pretender uma linguagem para troca de informação sobre a Internet. Outra grande vantagem é o bom nível de compressão (por exemplo, compressão zip) que o XML apresenta, poupando espaço e tempo na transmissão de informação entre as aplicações, que por vezes é em grandes quantidades.

Existem também alguns aspectos na descrição de objectos em XML que o XMI contempla:

- A representação dos objectos é feita através de elementos e atributos XML.
- Como os objectos estão normalmente interligados, o XMI inclui mecanismos *standard* para referenciar objectos no mesmo ficheiro ou noutros ficheiros.
- A identificação dos objectos permite que sejam referenciados a partir de outros objectos em termos ID's e UUID's.
- A versão dos objectos e das suas definições são também suportadas pelo XMI.

A usa criação resolveu alguns problemas de interligação de tecnologias (aplicações) e representação de informação. Desta forma, qualquer representação de informação em UML ou MOF, é também

possível em XMI. O uso do XMI permite que os programadores passam criar aplicações num ambiente de equipa com uma maior interoperabilidade e consistência. É possível colaborar em ambientes de desenvolvimento para criar, armazenar, trocar modelos na Internet, independentemente da plataforma, linguagem de programação ou ferramenta.

Cada documento XMI contem elementos gerados do modelo de dados da aplicação, mais um conjunto fixo de elementos que pertencem à especificação XMI. Este conjunto fixo de elementos definem a estrutura do documento e proporcionam um conjunto de tipos de dados. Essa estrutura começa com um elemento de topo, o elemento `XMI`. Os seguintes elementos fazem do elemento `XMI`:

- **Header**. Contem declarações das versões e documentações opcionais referentes à informação do documento.
- **Content**. Contem a informação do modelo de dados da aplicação.
- **Differences**. Especifica as diferenças entre dois documentos XMI gerados. É útil quando são feitas pequenas alterações em documento muito grandes.
- **Extensions**. Permite definir informação dependente da ferramenta. Por exemplo, no caso do UML, informação da representação gráfica dos elementos dos modelo, que pode ser diferente de ferramenta para ferramenta.

### 2.2.7 Processamento de documentos

Os programadores que usam XML podem tirar partido de bibliotecas de manipulação especializadas para incorporar funcionalidades como análise, validação e modificação de documentos. Essas bibliotecas designam-se por API (*Application Programming Interface*).

Um das vantagens na utilização de API's para trabalhar com documentos XML é a confirmação da boa formação dos documentos. Antes de se começar a usar as funcionalidades de uma API para XML é verificada a boa formação do documento. A simplicidade de manipulação do documento é outra das grandes vantagens. A manipulação de fragmentos do documento XML torna-se uma tarefa muito simples usando API's, já que elas dispõem de funcionalidades próprias para essa tarefa.

As API's para XML mais conhecidas são o DOM (*Document Object Model*) e o SAX (*Simple API for XML*). Estas API's são apresentadas em seguida enumerando as vantagens na utilização de cada uma.

#### DOM

O DOM é uma API independente da linguagem e da plataforma que permite aceder e modificar dinamicamente o conteúdo, a estrutura e o estilo de um documento XML[2]. O DOM é uma especificação, não uma implementação. Esta fica ao cargo das empresas de desenvolvimento de software[Fes01, DOM].

[2]A escolha da linguagem e da plataforma é, contudo, dependente do *parser* que for utilizado para expor a implementação DOM do documento XML

Um *parser* que use o DOM cria uma representação do documento completo na memória da máquina como uma árvore abstracta de nós. Como a representação total do documento permanece em memória, durante todo o processamento, os acessos a partes do mesmo são muito rápidas. Contudo, isto pode-se tonar numa grande desvantagem, pois se o ficheiro XML for de 10MB, com referências cruzadas, é impensável criar uma representação total do documento em memória, já que a tarefa pode ser bastante demorada e os recursos físicos podem-se esgotar.

Existem vários níveis de especificações do DOM. O nível 0 foi o primeiro a ser lançado e tinha definidas funcionalidades idênticas à do Netscape Navigator 3.0 e Internet 3.0. Em Outubro de 1998, foi lançada a recomendação DOM nível 1 que proporciona funcionalidades de manipulação da estrutura e conteúdo de documento XML e HTML. O nível 2 foi lançado em Novembro de 2000 adicionando funcionalidades à especificação do nível 1, tais como: interfaces que possibilitam a criação um documento, importação de nós de um documento para outro, suporte para *namespaces*, associação de folhas de estilos aos documentos, uso de modelo de objectos dos CSS (*Cascading Style Sheets*), filtros e iteradores e modelo de objectos por eventos. Actualmente existe o nível 3 do DOM como *Working Draft*. Nele foram adicionados funcionalidades para identificar unicamente nós num documento, comparar a posição relativa entre dois nós de um documento e a possibilidade de copiar nós de um documento para outro.

### SAX

Os programadores que comecem a utilizar uma API para trabalhar com documentos XML, têm alguma dificuldade em escolher a API que mais se adequa às suas necessidades. Pelo facto de muitos deles não terem prática nem conhecimento profundo das API's existente, o DOM acaba por ser a opção inicial, já que aparentemente é a mais simples de usar. Sabendo-se que com o DOM é possível fazer quase tudo a um documento XML, qual será a razão para o aparecimento de outra API? A razão é apenas uma: desempenho. O *Simple API for XML* (SAX) é um API *standard* com o objectivo de obter elevado desempenho no processamento de documentos XML. Consegue ser duas vezes mais rápido do que o DOM e utiliza muito menos memória[SAX].

Ao contrário do DOM, o SAX não é uma recomendação W3C. É software de domínio publico, criado por membros da *XML-DEV mailing list*, liderada por David Megginson. A implementação SAX original foi lançada em Maio de 1998, e foi substituída pela versão 2.0 em Maio de 2000. Inicialmente foi desenvolvida para Java, mas hoje em dia é suportada por muitas outras linguagens.

A razão de tão grande diferença de desempenho entre o uso de DOM e SAX é simples de compreender, bastando para isso verificar a organização de ambos. Normalmente, os *parsers* DOM carregam todo o documento XML para a memória através de uma estrutura de árvore, que proporciona acesso aleatório aos nós do documento e permite pesquisas no documento[3]. Caso se esteja a trabalhar com um ficheiro XML de 10MB, a última coisa que se pretende fazer é carregar todo o documento para a memória. Esta tarefa, não só, iria demorar algum tempo, como também iria ocupar a memória da máquina num ápice. O SAX resolve estes problemas, libertando a informação que já foi processada, para que apenas uma pequena porção de informação esteja em memória em cada instante.

Para atingir esses objectivos, o SAX usa um processamento baseado em eventos. O *parser* SAX lê do documento XML e notifica a aplicação os eventos importantes. Por exemplo, o *parser* pode ler

[3]Esta funcionalidade é apenas implementada por algumas aplicações.

a etiqueta de um elemento do documento, notificar a aplicação através do evento `startElement`. Depois de completo o evento, o *parser* não necessita de manter a etiqueta do elemento e informação relacionada em memória.

Os *parsers* SAX usam um conjunto predefinido de chamadas para notificar uma aplicação de eventos de *parsing*. Estas chamadas são definidas como métodos, de vários interfaces SAX *standard*, para serem implementados pelas aplicações.

#### Que modelo escolher?

Muitos programas não necessitam das funcionalidades proporcionadas pelo DOM. O uso de DOM pode originar problemas como a falta de memória, e pode diminuir dramaticamente o desempenho do processamento quando são manipulados documentos muito grandes[Bea01].

Geralmente, para processamentos rápidos e pouco intensivos, o SAX é a melhor escolha. Quando não é necessária uma manipulação do tipo de árvore, também deve ser usado o SAX. Para os restantes casos, o DOM proporciona um interface mais simples de usar.

As razões para o uso do SAX são as seguintes:

- **Uso de documentos grandes.** A grande característica da especificação SAX é a capacidade de processamento de documentos XML com *gigabytes* sem ultrapassar os recursos físicos existentes, porque não cria uma representação do documento completo em memória, como faz o DOM. Se os recursos forem poucos, deve ser usado o SAX.
- **Retirar um valor especifico de um documento.** É possível abortar um processamento SAX durante qualquer tratamento de evento. Ou seja, quando se obtém o que se pretende do documento, pode-se simplesmente abortar o processamento.
- **Criação de um sub-conjunto do documento.** Quando se lida com documentos grandes, o SAX pode ser usado para criar um sub-conjunto da informação pretendida, que pode depois ser processada com o DOM.

As razões para o uso do DOM são as seguintes:

- **Modificar o documento.** Como o SAX só lê documentos, não os consegue alterar. É claro que o SAX pode sempre criar novos documentos com as alterações, mas o DOM proporciona funcionalidades mais simples de usar já que permite modificar directamente o documento.
- **Acesso aleatório.** O DOM é mais adequado para aceder ao documento XML aleatoriamente, já que mantém todo o documento em memória.

## 2.3 Serviços Web

Os serviços Web são um novo tipo de aplicações Web. São aplicações que podem ser publicadas, localizadas e invocadas pela Web. Disponibilizam funcionalidades, desde tarefas muito simples até processos complicados. Depois de um serviço Web ser disponibilizado, pode ser invocado por outras

aplicações e até mesmo por outros serviços. A sua introdução veio permitir estabelecer ligações entre aplicações distribuídas globalmente, com o objectivo de transmitir e integrar grandes quantidades de informação de uma forma eficiente e com custos mais reduzidos.

Devido às limitações apresentadas por tecnologias como o CORBA e o OLE na área da computação distribuída, surgiu a necessidade de construir uma solução que fosse independente do vendedor, da plataforma e da linguagem. Tinha que oferecer mais do que uma plataforma que permita interoperabilidade. A implementação e publicação das funcionalidades, por parte dos programadores, tinha que ser mais simples. A solução tinha que passar por definir um modelo de computação distribuída baseado em *standards* Internet abertos.

Para tentar resolver os problemas inerentes à comunicação de informação entre aplicação, foi criada a noção de serviço Web, que define um conjunto de funções, agrupadas com uma única entidade publicada na rede, para serem usadas por outras aplicações. Através de uma mensagem, criada como um documento XML, uma aplicação envia um pedido a um serviço Web, disponível na rede, e, opcionalmente, recebe uma resposta, também na forma de um documento XML. Associados aos serviços Web, existem *standards* que definem o formato de uma mensagem, especificam o interface para onde a mensagem é enviada, descrevem as convenções do mapeamento do conteúdo da mensagem de e para o programa que implementa o serviço, e definem mecanismos de publicação e procura dos interfaces dos serviços Web.

Os serviços Web são o núcleo da nova geração de sistema distribuídos pelas seguintes razões:

- **Interoperabilidade.** Qualquer serviço Web pode interagir com outros serviços Web. Graças ao SOAP, o novo protocolo *standard* suportando pela grande maioria dos vendedores. A dificuldade de conversão de dados entre protocolos como o CORBA, DCOM, entre outros, desaparece.
- **Ubiquidade.** Os serviços Web comunicam XML através dos protocolos de transporte, como o HTTP e o SMTP. Qualquer periférico que suporte estas tecnologias pode implementar ou aceder a serviços Web.
- **Implementação simples.** Os conceitos por detrás dos serviços Web são fáceis de entender, e existem ferramentas do domínio público que permitem criar e disponibilizar serviços Web de uma forma simples e rápida. Pelo facto dos serviços Web poderem ser escritos em qualquer linguagem, os programadores não necessitam de alterar o ambiente de desenvolvimento, para criar ou usar serviços Web.
- **Suporte Industrial.** Todos os grandes fabricantes na área das tecnologias de informação suportam SOAP, que é um dos *standards* usados na implementação de serviços Web.

Por vezes, os serviços Web, são rotulados como soluções para problemas contemporâneos, mais propriamente na área da computação distribuída, ocupando o papel de tecnologias anteriores. No entanto, não vêm solucionar todos os problemas apresentados pelas tecnologias anteriores. Eles apenas introduzem um novo nível de tecnologia - outra forma de comunicação entre programas - mas não vêm substituir totalmente as tecnologias anteriores. Este novo nível de tecnologia permite novas funcionalidades, mas o mais importante, é que permitir a integração de mecanismos definidos a um nível de abstracção superior.

O aparecimento de novas tecnologias como os serviços Web, com novas potencialidades, não substitui totalmente as anteriores. Isto é, dependo dos objectivos que pretendemos atingir ou até o âmbito do sistema a implementar, a utilização de tecnologias mais antigas pode ser vantajosa. Por exemplo, o OLE foi adoptado pela Mirosoft originando o COM e o DCOM. Surgiram novas tecnologias, mas estas continuam a ser as utilizadas para que os documentos de uma aplicação possam incorporar documentos de outras. O mesmo acontece com o CORBA. Existem ambientes específicos, em a sua utilização é vantajosa, como por exemplo o uso num sistema com apenas uma máquina.

Como os interfaces Web, os serviços Web, são complementos das aplicações, programas, e bases de dados. O desenvolvimento de aplicações continua a necessitar do Java, do VB, do C. A única coisa que se altera é a forma de formatar a informação e de a comunicar.

Os serviços Web requerem algumas tecnologias baseadas em XML para permitir a comunicação entre as aplicações:

- **XML**. É a base em que os serviços Web assentam. É uma linguagem que permite definir o formato dos dados e a forma de os processar.

- **WSDL** (***Web Services Description Language***). Define os interfaces dos serviços Web, os tipos de dados, os tipos de mensagens, encaixes de interacções, e protocolos de mapeamento.

- **SOAP** ***Simple Object Access Protocol***. Define o formato dos documentos XML que usados na comunicação com serviços Web.

- **UDDI** (***Universal Description, Descovery, and Integration***). Define o registo e procura de serviços Web.

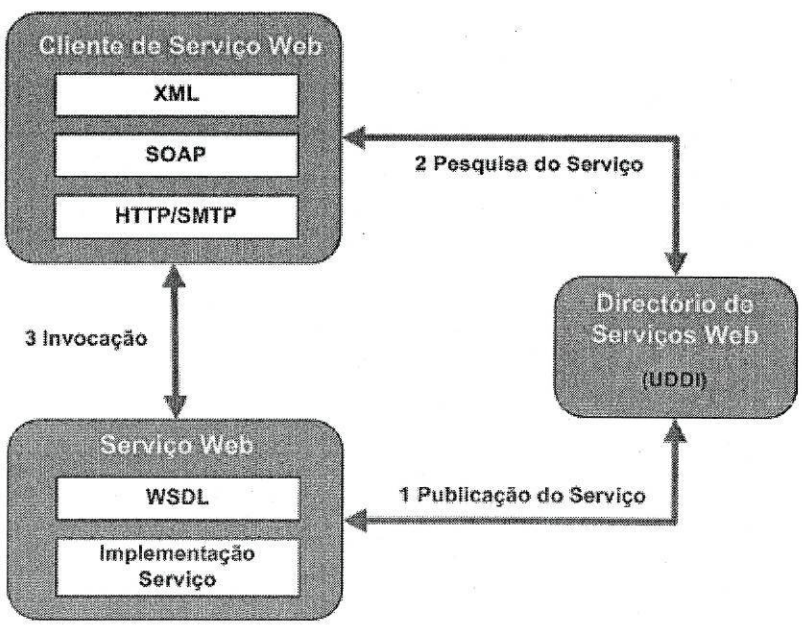



Figura 2.13: Tecnologias que rodeiam os serviços Web.

### 2.3.1 Comunicação entre aplicações

A Web foi desenhada para permitir a interacção humana com documentos em hipertexto. Contudo, a Web não permite uma boa interacção entre aplicações, nomeadamente na transferência e integração de grandes quantidades de informação. As tendências como a diminuição do preço e aumento da capacidade do armazenamento e da largura de banda, o uso de conteúdos dinâmicos e o uso dos mais diversos periféricos, com acessos de diferentes plataformas, realça ainda mais a necessidade de uma tecnologia para solucionar estes problemas. É necessário um método mais eficiente, que permita que as aplicações, interajam directamente umas com as outras[New02, Tid00, Bea01].

Nas últimas duas décadas as redes de computadores têm ganho cada vez mais destaque, e a comunicação entre computadores ganhou importância. Os modelos de objectos, como o COM/OLE (*Component Object Mode/Object Linking Embedding*) e o CORBA (*Common Object Request Broker Architecture*), foram extendidos para suportarem comunicações através de redes. A Microsoft apresentou o DCOM (*Distributed COM*) como sendo o seu protocolo RPC (*Remote Procedure Call*), permitindo ultrapassar a limitação de uma só máquina, e a comunidade Java estabeleceu o IIOP (*Internet Inter-ORB Protocol*) como o protocolo CORBA *standard*. Outro grande concorrente destes é o protocolo RMI (*Remote Method Invocation*) da Sun Microsystem's. O RMI faz de ponte sobre o IIOP, como forma das aplicações Java que usam RMI se conectarem ao CORBA.

Para implementar um sistema eficiente de comunicação entre aplicações de diferentes máquinas, as alternativas eram o DCOM, CORBA e o RMI. Contudo, implementar comunicações entre aplicações usando estes protocolos tornava-se uma tarefa bastante complexa. Tornava-se bastante difícil desenvolver um sistema devido aos erros de instalação, configuração e administração. Havendo requisitos adicionais de controlo de transacções, a instalação e configuração tornavam-se ainda mais complicadas.

Outro grande obstáculo é a comunicação numa rede heterogénea como a Internet, ao ser impossível garantir que ambas as partes da comunicação implementem o mesmo modelo de distribuição de objectos. Os protocolos mencionados centravam-se em soluções proprietárias, para tirarem o máximo partido do protocolo. Existiram algumas implementações desses protocolos em sistemas operativos e linguagens diferentes daquelas que tinham sido desenvolvidas, mas sem grande sucesso. Por exemplo, a primeira linguagem de implementação do CORBA foi o Java, que teve bastante aceitação, enquanto que as implementações noutras linguagens foram insignificantes. O uso destes protocolos tinha ainda um problema mais grave que era a dificuldade de funcionamento sobre *firewalls* ou servidores *proxy*. Foram criadas algumas ferramentas para contornar estes problemas, que apenas existiam em algumas linguagens e tornavam a comunicação mais lenta.

As capacidades da Web como distribuidora de informação, a simplicidade de acesso por nomes e a ubiquidade eliminam os problemas das plataformas de *midleware* em que a interoperabilidade é difícil de atingir. A Web proporciona um extenso e uniforme interface de acesso e ligação entre os serviços tornando mais fácil a implementação em relação às plataformas existentes.

Em seguida é apresentado o CORBA e o OLE para se ter uma ideia do funcionamento das plataformas antes de se usar os serviços Web.

### CORBA

O CORBA é o resultado de uma tentativa de normalização, por parte de múltiplas empresas reunidas no consórcio OMG, para permitir interoperabilidade entre nós computacionalmente distribuídos. Com o CORBA pretende-se criar uma ambiente genérico de desenvolvimento de aplicações baseadas em objectos. O CORBA define uma arquitectura para objectos distribuídos. O seu paradigma principal é fazer pedidos de serviços de objectos. Tudo o resto é baseado neste paradigma[COR].

Os serviços de um objecto são proporcionados pelo seu interface. Os interfaces são definidos pela IDL (*Interface Definition Language*) do OMG. A IDL é uma linguagem declarativa e corresponde a um subconjunto do C++, a que foram adicionadas algumas palavras-chave para suportar mecanismo de ligação às linguagens de programação mais utilizadas. Um cliente acede a um objecto efectuando um pedido desse objecto. Um pedido é um evento, em que a sua informação incluí uma operação, uma referência ao objecto que proporciona o serviço, e possivelmente outros parâmetros. A referência ao objecto corresponde ao seu nome[Ins99, Gis01, MG98, OH97].

A componente central do CORBA é o ORB (*Object Request Broker*). O ORB é o sistema distribuído a que os clientes recorrem para fazer os pedidos aos objectos remotos. Os principais pontos que caracterizam um ORB são:

- Invocação estática e dinâmica de objectos.
- Independência da linguagem de alto nível, suportando múltiplos formatos de invocação das linguagens mais utilizadas.
- Inventário das interfaces disponíveis. Um serviço de registo de interfaces que permite dinamicamente conhecer os métodos, respectivos parâmetros, etc.
- Mensagens polimorficas. O ORB não invoca apenas uma função remota, invoca sim uma função no objecto alvo. Isto quer dizer que o efeito de chamar a mesma função várias vezes pode ter diferentes resultados, dependendo do objecto que executa a função.
- Segurança.
- Suporte para ligação a monitores transaccionais.

O ORB recebe uma mensagem de um cliente, para invocar um método especifico sobre um objecto registado. Este interpreta a mensagem e procura um objecto que responda ao pedido, envia-lhe os parâmetros, invoca o método deste, e retorna os resultados obtidos. O nó do cliente, não precisa de se preocupar em saber onde é que o objecto está, qual é a linguagem de programação, qual é o sistema operativo, ou outros aspectos do sistema que não fazem parte do interface do objecto.

A figura 2.14 exemplifica um pedido de um serviço de um objecto segundo o CORBA.

Para permitir que diferentes implementações de ORB's comuniquem, o formato das mensagens tem de estar bem definido. O CORBA usa o protocolo GIOP (*General Inter-ORB Protocol*) para definir o formato dessas mensagens. Cada mensagem contem informação à cerca da operação a realizar, os parâmetros e a *byte order*, caso seja necessário. Os parâmetros e os valores de retorno são enviados nas mensagens GIOP segundo o protocolo CDR (*Common Data Representation Protocol*). Este

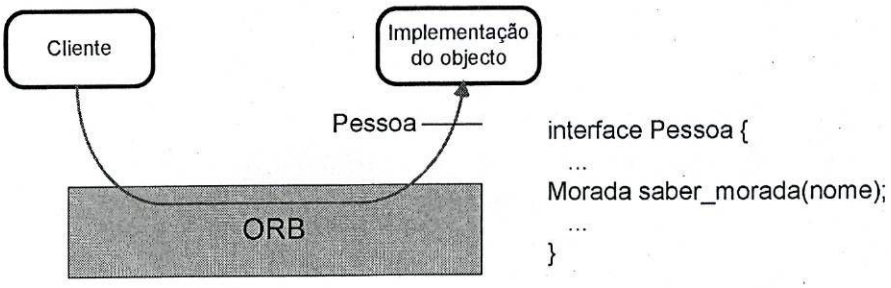



Figura 2.14: Funcionamento de um pedido CORBA.

protocolo define o processo de representação das primitivas IDL em formato de texto. Qualquer aplicação CORBA permite receber e entender dados no formato CDR[Ols].

O IIOP (*Internet Inter-ORB Protocol*) define a camada de transporte de envio de mensagens GIOP sobre o TCP/IP. O IIOP faz o mapeamento das mensagens GIOP para mensagens TCP/IP e faz de protocolo de comunicação quando se pretende enviar mensagens GIOP pela Internet.

A criação do CORBA trouxe grandes benefícios na área dos sistemas distribuídos. Contudo, é uma tecnologia que sempre teve algumas dificuldades para conseguir cativar utilizadores. Um dos motivos é a complexidade de instalação, configuração e administração de aplicações CORBA. Outro motivo é o facto de existirem poucas implementação de ORB's completos gratuitas. A dificuldade que existe em conseguir usar o IIOP sobre *firewalls* e servidores de *proxy* é outro dos motivos. O aparecimento de tecnologias, em parte concorrentes, menos complexas e com outros recursos, também veio contribuir para a pouca aderência ao CORBA[dJea02].

**OLE**

O OLE é uma tecnologia que permite que aplicações criem documentos em que o conteúdo provem de diversas fontes. Por exemplo, um processador de texto que use esta tecnologia, permite criar documentos que pode incorporar objectos do tipo folha de cálculo. Ao contrário do método tradicional de "cortar-e-colocar" em que a aplicação receptora transforma o formato da informação que recebe, os documentos embutidos mantêm todas as suas propriedades. Caso se pretenda alterar o documento embutido, é activada a aplicação responsável por esse documento, para que a alteração possa ser feita[OLE].

A partilha de informação pode ser feita através de ligação ou incorporação de objectos. Na partilha segundo a ligação de objectos a informação que é partilhada é a original, enquanto que na incorporação de objectos é feita uma cópia da informação original. Caso de altere um objecto ligado, a informação é alterada em todas as aplicações, enquanto que na incorporação apenas é alterada a informação na aplicação individual.

O modelo de transferência de dados do OLE permite que os utilizadores transfiram informação através de uma operação *drag and drop*, copia-cola, ou programando. O modelo é suportando pelo IDataObject. A implementação do IDataObject permite que a informação seja colocada em vários formatos. Desta forma as aplicações têm de recorrer a métodos do IDataObject para pedirem a informação no formato pretendido.

### 2.3.2 Descrição de serviços

O WSDL *Web Services Definition Language* foi criado para representar a definição de serviços Web, por parte da Ariba, IBM e Microsoft. Em Março de 2001, o WSDL recebeu o estatuto de nota pelo W3C, significando que ainda não era uma especificação completa, mas que iria ser considerado pelo W3C e pelos vendedores. A grande maioria das implementações de serviços SOAP, passou a usar o WSDL[Lan02, Bea01, WSD].

O WSDL é uma linguagem XML, que permite descrever serviços Web. A descrição dos serviços abrange a definição do serviço propriamente dito, o que é que ele faz, onde é que ele reside, e a forma de o invocar. No WSDL, os serviços são vistos como conjuntos de pontos na rede ou *portas.*

No WSDL, a definição abstracta das portas e das mensagens é separada do seu endereço na rede e da associação do formato dos dados. Isto permite re-usar definições abstractas das mensagens, que são descrições da informação que vai ser transmitida, e dos tipos de portas, que são conjuntos de operações. Uma porta é definida associando um endereço de rede a uma associação (que pode ser utilizada múltiplas vezes); uma colecção de portas define um serviço. Desta forma, um documento WSDL usa os seguintes elementos na definição dos serviços como parte do elemento `definitions`[Vas01]:

- **Types**. Um repositório de definição de tipos de dados usando uma especificação XML (por exemplo, XSD).
- **Message**. Definição abstracta das mensagens usadas pelos serviços através dos tipos de dados existentes.
- **Port Type**. Conjunto abstracto de operações suportadas por um ou mais pontos, em que cada operação pode ter definida uma mensagem de entrada e outra de saída.
- **Binding(associação)**. Especificação do protocolo de transporte (HTTP,SMTP) e da forma como são comunicadas as operações e as respectivas mensagens.
- **Port**. Um simples ponto definido como a combinação de uma associação e de um endereço de rede.
- **Service**. Uma colecção de pontos relacionados.

O WSDL é extensível para permitir descrever serviços Web independentemente dos protocolos de rede e da mensagem. A especificação actual (1.2) permite usar associações para o SOAP 1.2, HTTP e MIME, e não incluí uma forma completa de descrever a composição e orquestração dos serviços[HTT, MIM].

### 2.3.3 Comunicação de dados

#### Introdução ao SOAP

O SOAP é um protocolo de comunicação para troca de informação entre computadores, independentemente do sistema operativo ou do ambiente de programação. Na sua especificação, é definido como sendo um protocolo que necessita de poucos recursos, para a troca de informação estruturada

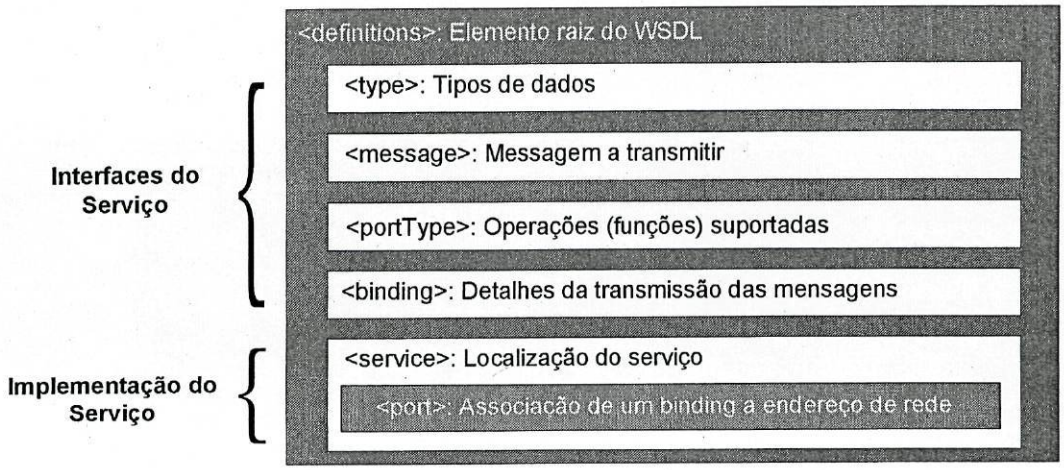



Figura 2.15: Estrutura de uma documento WSDL.

e tipada, entre máquinas, num ambiente distribuído e descentralizado. Estas últimas caracterizações referem-se, normalmente, a redes como a Internet, mas a especificação não refere que o SOAP apenas pode ser utilizado neste tipos de redes. Por exemplo, o SOAP pode ser usado em comunicações entre computadores numa intranet. O SOAP é um protocolo baseado em XML, que não define um modelo de programação ou uma semântica especifica (por exemplo, uma API). Apenas define um mecanismo simples de troca de mensagens, proporcionando um modelo de empacotamento modular e uma forma de codificação nas mensagens[Bea01, Vas01, SOA].

O SOAP pode ser visto como um protocolo de transporte para envio de mensagens. De mesma forma que o correio postal tem de seguir algumas especificações para envio das cartas, o SOAP assume necessidades similares em comunicações entre computadores. Por exemplo, o correio postal envia as cartas dentro do respectivo envelope, enquanto que o SOAP define um envelope para a informação a transmitir.

É importante lembrar que o SOAP não define um protocolo de transporte para envio de mensagens. Seguindo a analogia do correio postal, um envelope não especifica se a carta é enviada por avião, comboio ou automóvel (mas os selos já dão essa informação). A especificação SOAP 1.1 descreve a forma como o HTTP deve ser usado para transportar mensagens SOAP, mas indica claramente que o SOAP não está limitado ao protocolo de transporte HTTP, pode ser usado através de outros, nomeadamente o SMTP, o protocolo de transporte de correio electrónico.

Ao suportar o SOAP, os servidores de HTTP podem proporcionar serviços Web acessíveis a outros computadores, sem a necessidade da interacção ou intermediação humana. Desta forma, uma aplicação pode reunir soluções a partir de qualquer combinação de componentes de *software*, aplicações, processos, ou periféricos. Por exemplo, um ASP (*Application Service Provider*) pode usar SOAP para disponibilizar uma solução genérica para autenticação de clientes, pagamentos, entre outras coisas.

### A necessidade de um novo protocolo

O protocolo SOAP não inventa coisas novas tecnologias, mas usa as existentes com o XML, HTTP, SMTP e outros protocolos Internet comuns. O XML define o formato das mensagens, do envelope, e do mecanismo RPC, e o HTTP, SMTP, ou outros protocolos Internet são usados como mecanismos de

transporte das mensagens. Por este facto, e ao contrário dos *standards* de comunicação estabelecidos, o SOAP é totalmente independente da plataforma e da linguagem.

Como o seu nome sugere, o SOAP é um protocolo simples. Com apenas algumas linhas de código, um *parser* XML, e um servidor HTTP, é possível ter um ORB SOAP - um sistema que possibilita a invocação de objectos, controlando o seu tempo de vida. Isto evita a complexidade dos outros protocolos. Pela sua extensibilidade, através do uso de XML, o SOAP garante que pode ser usado em qualquer plataforma e para qualquer situação.

O SOAP trabalha sobre a infra-estrutura da Internet. Não é necessária nenhuma configuração especial dos *routers* ou servidores *proxy* para ter o SOAP a funcionar. Usando o HTTP, ou outro protocolo de transporte da Internet, não é necessária nenhuma modificação de *firewalls*. O SOAP propórciona um ambiente de comunicação, mas não limita o uso do protocolo de transporte, tornando-o extensível também a este nível.

O reconhecimento do SOAP pelo W3C garante que nenhum vendedor tem o controlo absoluto sobre o SOAP, tornando-o um *standard* aberto, como o HTML, XML, ou HTTP e SMTP.

### Mensagem SOAP

As mensagens SOAP são fundamentalmente transmissões unidireccionais, do emissor para o receptor, normalmente implementando comunicações cliente/servidor. Todas as mensagens SOAP são documentos XML, com os respectivos *schemas*; incluem *namespaces* próprios em todos os elementos e todos os atributos. O SOAP define dois *namespaces*, envelope SOAP e seriação ou codificação SOAP. Algumas das regras que o SOAP especifica são mais estritas que o XML por si só, por exemplo, as mensagens SOAP não podem conter um DTD ou instruções de processamento. Isto não é surpresa, pois um dos objectivos do SOAP é ser um protocolo simples.

As mensagens SOAP são documentos XML, constituídas por três secções, representadas na figura 2.16:

- **Envelope**. É o elemento de topo do documento XML que representa a mensagem.
- **Cabeçalho**. É um mecanismo genérico e opcional para adicionar características às mensagens SOAP.
- **Corpo**. É a secção que contem a informação propriamente dita a ser transmitida.

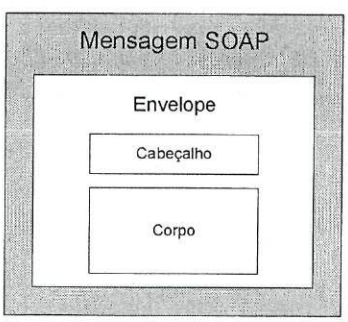



Figura 2.16: Estrutura da mensagem SOAP.

```
1 POST /objectURI HTTP/1.1 Host: www.foo.com SOAPMethodName:
2 urn:develop-com:IBank#getBalance Content-Type: text/xml
3 Content-Length: nnnn
```

Figura 2.17: Exemplo de cabeçalho HTTP.

### Interpretação de uma mensagem SOAP

O SOAP permite que sejam invocados métodos entre pontos do HTTP. Um ponto SOAP é identificado por um URL (como um qualquer recurso HTTP). Um método SOAP é identificado unicamente por um URI e por um nome simbólico do método. Os pedidos de métodos SOAP são transportados em pedidos HTTP POST. O cabeçalho HTTP, deve ter identificado o método a ser invocado. A figura 2.17 apresenta um exemplo de um cabeçalho HTTP.

Este cabeçalho HTTP indica que o método *getBalance* deve ser invocado no ponto identificado por `http://www.foo.com/objectURI`.

O conteúdo HTTP da invocação de um método SOAP é um documento XML, que contem a informação necessária ao processo. Assumindo que a única coisa necessária para obter o saldo de uma conta é o número de conta, o documento XML, da mensagem HTTP respectiva, deve ser algo idêntico ao apresentado no documento 2.21.

```
1  <?xml version='1'?>
2  <SOAP:Envelope
3      xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1'>
4      <SOAP:Body>
5          <i:getBalance
6              xmlns:i='urn:develop-com:IBank'>
7              <account>23619-22A</account>
8          </i:getBalance>
9      </SOAP:Body>
10 </SOAP:Envelope>
```

Programa 2.21: Documento SOAP no corpo de uma mensagem HTTP.

Observando o conteúdo do elemento `SOAP:Body`, o elemento de topo é o nome do método a invocado, que corresponde ao nome do cabeçalho HTTP. Esta redundância existe para permitir que infra-estruturas HTTP (*proxies*, *firewalls*, servidores Web) processem a chamada sem fazer a verificação sintáctica do XML, mantendo também a mensagem XML independente da mensagem HTTP que a envolve.

Depois de recebido o pedido do lado do servidor, é executado o código que corresponde ao método invocado. A forma de execução está completamente fora do âmbito do protocolo SOAP. As reacções à invocação de pedidos incluem a execução de:

- *CGI.*
- Um módulo *Apache.*
- Página *PHP* ou *ASP.*
- *Servlet* Java.
- Folha de estilos XSLT.

Depois de criada a resposta ao pedido, uma mensagem HTTP vai ser retornada para o cliente, com os resultados da operação. Não existem cabeçalhos HTTP de resposta específicos para o SOAP. Contudo, o corpo da mensagem HTTP contem um documento XML, correspondendo à mensagem SOAP de resposta. Um pacote HTTP da transmissão de uma mensagem SOAP é apresentado na figura2.22.

```
1  200 OK Content-Type: text/xml Content-Length: 265
2
3  <?xml version='1.1'?>
4  <SOAP:Envelope
5     xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1'>
6     <SOAP:Body>
7        <i:getBalanceResponse
8           xmlns:i='urn:develop-com:IBank'>
9           <amount>45.21</amount>
10       </i:getBalanceResponse>
11    </SOAP:Body>
12 </SOAP:Envelope>
```

Programa 2.22: Pacote HTTP na transmissão de uma mensagem SOAP.

### Relação do SOAP com o WSDL

O WSDL é uma especificação que descreve serviços Web segundo a gramática comum XML. É independente da linguagem e da plataforma e é usado principalmente (mas não exclusivamente) para descrever serviços SOAP.

Esta especificação descreve quatro componentes de informação cruciais, para a definição de serviços SOAP:

- Funções disponíveis.
- Tipo de dados das mensagens dos pedidos e das respostas.
- Protocolo de transporte a ser utilizado.
- Endereços de localização dos serviços

O uso do WSDL, permite documentar os serviços Web, facilitando o trabalho dos clientes na localização dos serviços e na invocação das funções disponibilizadas. Com o WSDL, o processo de adição de novos serviços a aplicações, através de ferramentas próprias, tornou-se quase automático. Desta forma, a utilização do SOAP como forma de comunicação entre aplicações, tem grandes vantagens no uso do WSDL.

### Relação do SOAP com os protocolos de comunicação (HTTP,SMTP)

Como já foi referido, o SOAP não dependente dum único protocolo de transporte. A especificação do SOAP descreve a forma como pode ser usado em combinação com o HTTP, a título de exemplo. O SOAP pode ser usado sobre outros protocolos, nomeadamente SMTP (*Simple Mail Transport Protocol*).

O SOAP veio acrescentar mais uma camada relativamente aos protocolos de transporte. Por exemplo, numa comunicação usando HTTP, existe alguma meta-informação no cabeçalho (apenas relativa ao protocolo de transporte) e o corpo da mensagem contem unicamente a mensagem propriamente dita. O SOAP tem uma estrutura idêntica, mas a outro nível. Para além de tirar partido de um protocolo de transporte deste tipo, permite organizar a informação do corpo da mensagem. Contem um cabeçalho que pode incorporar meta-informação num formato XML à escolha, e o corpo da mensagem é estruturado segundo um *XML Schema*.

#### Comunicação síncrona usando HTTP

O HTTP, sendo um protocolo de transporte da Internet é ubíquo sobre a Web - grande parte da informação da Web é disponibilizada segundo o HTTP. As *firewalls* facilitam o uso do HTTP, permitindo, na maioria dos cados, a passagem de informação segundo este protocolo. O HTTP também pode ser usado juntamente com o SSL (*Secure Socket Layers*) certificando-se de forma a tornar-se HTTPS (*HTTP Secure*). Desta forma, todo o tráfego pode ser codificado entre os dois pontos da rede. Quando o SOAP usa o HTTP como protocolo de transporte, tem disponíveis todas as características do HTTP, como a sua ubiquidade.

O SOAP usa o modelo pedido/resposta segundo o HTTP de uma forma intuitiva, particularmente em comunicações do tipo de invocação remota de procedimentos. O HTTP dispõem de dois tipos importantes de pedidos: GET e POST. O método GET coloca toda a informação HTTP atrás do URL, e o POST incorpora toda a informação HTTP no cabeçalho do pedido. A especificação do SOAP dispõem de detalhes para o uso do método POST.

#### Comunicação assíncrona usando SMTP

O HTTP é a escolha natural para o uso do SOAP. Contudo, o protocolo Internet de transporte que deve ser usado para transferências de mensagens assíncronas ou de apenas uma mensagem, é o SMTP.

Para um cenário de perspectiva de negócio, é impossível usar sempre comunicação síncrona através do HTTP. Por exemplo, se o sistema for lento, o sistema tem de fazer muitos cálculos antes de submeter a resposta ao pedido, ou o sistema tem de simplesmente esperar por uma intervenção humana. Nestes casos, não faz sentido esperar por uma resposta, mas seria útil realizar a comunicação de forma assíncrona.

### 2.3.4 Segurança

Nos dias de hoje, todos os sistemas de informação têm preocupações de segurança. O mesmo acontece com os sistemas de transferência de informação. A informação transferida é privada, isto é, só pode estar acessível ao organismo emissor e ao receptor.

Os serviços Web não são excepção. Por exemplo, no serviço Web de uma livraria digital, quando o cliente envia informação privada (número do cartão de crédito) para o serviço de modo a efectuar uma compra, esta transferência tem de estar protegida contra intrusos.

Esta secção começa por questionar a segurança do SOAP. Depois aborda algumas questões de segurança dos serviços Web. Seguidamente apresenta algumas soluções, que podem ser usadas nos serviços Web. Por último descreve uma solução que foi testada no projecto.

### Considerações de segurança

Desde a publicação da especificação SOAP que se tem dado especial atenção a questões relacionadas com a segurança, nomeadamente a autenticação. Mesmo considerando estas questões como fundamentais, a especificação SOAP não incorpora nada que garanta a segurança da informação. Mas então, qual a razão do SOAP ignorar tais aspectos[Vas00]?

A resposta é simples: não faz parte do seu trabalho.

Questões relacionadas com a segurança não fazem parte dos objectivos que foram definidos para a especificação do SOAP. O SOAP é um protocolo de alto nível, que é sempre usado em conjunto com um protocolo de transporte. É extensível, permitindo assim trabalhar com outras especificações. Assenta na utilização de *standards* para tirar o máximo partido da infra-estrutura da Internet.

As considerações de segurança podem ser implementadas ao nível dos protocolos de transporte, ou então através do uso de especificações como extensões. Algumas delas são descritas mais a baixo, mas antes são identificados os requisitos de segurança.

### Requisitos de segurança

Existe um conjunto de requisitos de segurança que devem ser considerados em qualquer sistema de comunicação de informação em que se incluem os serviços Web. A tabela 2.6 apresentada esses requisitos, bem como soluções para os mesmos. Mais à frente será feita uma descrição mais detalhada de cada um dos requisitos[4][Hou01]:

| Requisito | Descrição | Solução |
|---|---|---|
| **Visibilidade Confidencialidade** | Os dados só podem ser visíveis pelos organismos pretendidos. | SSL |
| **Autorização** | O emissor dos dados tem de ter autorização para os enviar. | MAC |
| **Integridade** | Os dados não são modificados durante a transmissão, acidentalmente ou propositadamente. | DSIG |
| **Autenticação** | Os dados são enviados por um emissor propriamente identificado. | DSIG, MAC |
| **Não Repudiação** | Um emissor de uma mensagem não tenha hipótese de negar o seu envio. | DSIG |

Tabela 2.6: Requisitos de segurança e as respectivas soluções.

Os requisitos de segurança enumerados, vão ser abordados durante a apresentação de algumas

[4]As soluções indicadas são apresentadas mais à frente.

soluções. A autenticação e a autorização vão ser destacados porque são os requisitos que mais vezes se garantem.

### Autenticação e Autorização

Um serviço só pode permitir o acesso às suas funcionalidades a determinados clientes. Para que isso aconteça, os clientes têm de passar por um processo de identificação antes de terem acesso ao serviço. Este processo engloba autenticação e autorização que são muitas vezes consideradas um único processo, mas que na realidade são questões independentes.

Em primeiro lugar é necessária uma autenticação do utilizador, para verificar a sua identidade. Depois de completa e confirmada a validade da identificação, o utilizador passa a estar autenticado no serviço. O uso das funcionalidades do serviço, por parte do utilizador, requer sempre uma verificação prévia da sua autorização.

Enquanto que a autenticação tem que ser sempre feita, para identificar o cliente, antes de se ter acesso ao serviço, a autorização refere-se às funcionalidades permitidas pelo serviço, isto é, sempre que o cliente pretenda usar usar uma funcionalidade é verifica a sua autorização para tal. As responsabilidades da autorização são da aplicação, não podendo ser feito qualquer controlo ao nível dos protocolos de transporte.

O SOAP permite implementar autenticação de duas formas: ao nível do protocolo de transporte ou então como uma extensão do próprio protocolo.

A autenticação pode ser feita sobre os seguintes aspectos:

- **Mensagem**. A autenticação da mensagem garante que a mensagem não foi alterada durante a sua transmissão. Garante também que a mensagem foi enviada pelo emissor correcto. Este tipo de autenticação pode ser feito através do uso de MAC's ou assinaturas digitais.
- **Cliente/Servidor**. Este tipo de autenticação garante que o emissor e o receptor da mensagem são quem dizem ser. Isto é, o emissor pode confirmar a identidade do receptor, e o receptor pode fazer o mesmo. Este tipo de autenticação nada garante à cerca de quem criou a mensagem. O SSL é uma das tecnologias que permite fazer este tipo de autenticação.

Quanto à autenticação de mensagens são usadas as seguintes técnicas[Sch96]:

- **MAC (*Message Authentication Code*)**. Um Mac é uma função *hash* dependente de uma chave. Como o MAC usa uma chave privada, que é partilhada pelo cliente e pelo servidor, é apenas garantido que a mensagem foi criada pelo emissor ou pelo receptor. Da perspectiva de um terceiro organismo, não é possível determinar qual das dois organismos é que criou a mensagem. Só quem tiver a chave é que pode verificar o *hash*. Os MAC's podem ser usados juntamente com o SSL ou com o SOAP-DSIG. Pelo facto de ser computada mais rapidamente do que a DSIG, é bastante usada ao nível dos protocolos de transporte.
- **DSIG (*Digital Signature*)**. As assinaturas digitais são usadas para identificar a origem das mensagens. É uma tecnologia que usa chaves públicas e privadas. Quem envia a mensagem usa a chave privada e o receptor verifica a mensagem com a chave pública. A possibilidade de assinar

as mensagens SOAP foi conseguida com o aparecimento do SOAP-DSIG. Esta especificação define o formato de dados XML, da assinatura, que é adicionada à mensagem SOAP. Pelo facto das assinaturas digitais serem baseadas em criptografia de chave pública, geralmente, têm uma computação mais demorada em relação ao MAC. O SOAP-SIG garante que quem criou a mensagem foi quem a assinou.

Algumas das técnicas de autenticação cliente/servidor mais usadas são feitas a um nível mais baixo, isto é, ao nível dos protocolos de transporte. Em seguida são apresentadas algumas formas de o fazer sobre HTTP, sendo depois apresentado o SSL.

### HTTP (*HyperText Transport Protocol*)

O HTTP é o protocolo de transporte mais usado com serviços Web. Existem várias formas de autenticação de clientes usando este protocolo[Sun01].

As especificações HTTP (RFC 2616 e RFC 2517) definem um mecanismo de autenticação designado por autenticação básica. Neste tipo de autenticação, é enviada pelo cliente uma credencial (por exemplo, nome e *password*) para o serviço no cabeçalho do pedido HTTP. O servidor HTTP pode autenticar o cliente ou então passar a credencial para o serviço Web e este executa essa tarefa.

Foi também definido um mecanismo de autenticação HTTP mais elaborado, designado por autenticação *Digest*. Esta tinha como objectivo melhorar o anterior, obrigando o cliente a criar um *hash* MD5 das credenciais a partir da chave enviada pelo servidor. Desta forma, o cliente envia o *hash*, em vez das credenciais. O funcionamento base é o mesmo do mecanismo anterior.

Outro mecanismo de autenticação, é a passagem das credenciais codificadas como parte do URL do pedido HTTP. Este mecanismo é menos seguro que os anteriores, contudo, permite uma maior flexibilidade na forma como as credenciais são processadas, antes de serem passadas para o serviço. O URL 2.23 ilustra este mecanismo.

```
http://.../teste?nome=xxx&password=yyy
```

Programa 2.23: Passagem de credenciais de segurança no URL.

Nenhum dos mecanismos apresentados é totalmente seguro. No primeiro, o cliente envia as credenciais para o serviço como texto simples. Isto é extremamente grave, do ponto de vista de segurança, já que torna as credenciais visíveis, a qualquer observador não autorizado. O segundo mecanismo resolve este problema através do envio de um *hash*, em vez de credenciais. Por outro lado, não protege o conteúdo da mensagem HTTP, já que o envia como texto simples. Isto torna visível a mensagem a observadores não autorizados, e permite alterações da mesma, antes dela chegar ao destino. O terceiro mecanismo tem o mesmo problema e ainda o problema das credenciais ficarem registadas nos *logs* dos serviços e dos proxies.

### SSL (*Secure Socket Layer*)

O TCP/IP (*Transmission Control Protocol/Internet Protocol*) controla o transporte de informação sobre a Internet. Outros protocolos, como o HTTP, LDAP (*Lightweight Directory Access Protocol*),

ou IMAP (*Internet Messaging Access Protocol*), situam-se a um nível mais elevado e usam o TCP/IP. Isto é, pertencem ao nível das aplicações, enquanto que o TCP/IP está ao nível da rede.

O SSL (*Secure Sockets Layer*) é um protocolo que funciona entre o nível das aplicações e o TCP/IP, permitindo criar um canal de comunicação seguro entre o cliente e o servidor. A criação desse canal é feita através de um processo de autenticação, originando um canal de comunicação cifrado.

A criação do canal seguro é feita em três passos:

- **Autenticação SSL do servidor**. Confirmação da identidade do servidor por parte do cliente. A aplicação SSL do cliente, pode usar técnicas *standard* de criptografia de chave publica para confirmar que o certificado do servidor e a sua identificação são válidas segundo uma CA (*Certificate Authority*) que o cliente confie. Esta confirmação pode ser importante se por exemplo, o utilizador pretender enviar o número do cartão de crédito pela rede e pretende verificar a identidade do servidor.
- **Autenticação SSL do cliente**. Confirmação da identidade do cliente por parte do servidor através das mesmas técnicas da autenticação anterior. Este tipo de confirmação pode ser importante se o servidor, por exemplo, é um banco que pretende enviar informação confidencial a um cliente e quer confirmar a identidade do receptor.
- **Ligação SSL cifrada**. Toda a informação enviada entre o cliente e o servidor é cifrada pela aplicação do emissor e decifrada pela aplicação do receptor.

O protocolo SSL inclui dois sub-protocolos: o protocolo de registos SSL e o protocolo *handshake* SSL. O primeiro define o formato usado para transmitir a informação. O segundo usa o primeiro para trocar uma série de mensagens entre o cliente e o servidor para estabelecer a ligação segura. Essa troca de mensagens tem como objectivo facilitar os passos de criação de um canal seguro entre o cliente e o servidor.

O SSL também tem desvantagens. O processo para cifrar e decifrar as mensagens é demorado, já que requer um trabalho intensivo do CPU. Algumas experiências foram feitas e revelam que existe um atraso no processamento de transacções SSL de cerca de 90 por cento.

A implementação de uma comunicação segura deve então ter em conta o tipo de informação que se pretende transmitir. Desta forma, pode-se dividir a informação em três tipos: a que necessita de protecção de visibilidade e alteração, a que só necessita de protecção contra alteração e a que não necessita de protecção. Para o primeiro tipo deve ser usado o SSL. O segundo tipo necessita apenas de usar o mecanismos como o MAC, retirando trabalho de CPU. O último tipo não necessita de mecanismos de protecção.

### Autenticação através de extensões

Os mecanismos de segurança podem também ser implementados a um nível superior, isto é, ao nível das aplicações. Para isso podem ser usados *standards* de segurança já existentes[5].

Equipas do W3C têm vindo a desenvolver, nos últimos anos, especificações XML que criam mecanismos de segurança sobre informação XML. O XML-SIG (*XML Digital Signature*) é um dos resultados

[5]Devido ao facto de existirem muitos *standards*, apenas serão abordados os XML, que estão no âmbito do projecto.

desse trabalho, definindo uma sintaxe e regras de processamento, para usar assinaturas digitais sobre qualquer informação XML. O XML-ENC (*XML Encryption*) é outro *standard* desenvolvido para cifrar/decifrar informação digital. Foi também desenvolvida uma sintaxe para representar a informação necessária para cifrar/decifrar. O XKMS (*XML Key Management Services*) especifica protocolos para distribuir e registar chaves publicas, que são posteriormente usadas no XML-SIG e no XML-ENC.

Este conjunto de *standards* pode ser usado sobre qualquer tipo de mensagem XML. Para se perceber melhor como estes *standards* facilitam a implementação de segurança, consideremos os passos da transmissão duma mensagem SOAP:

- O emissor da mensagem especifica o destinatário no cabeçalho da mensagem SOAP.
- O emissor utiliza o XKMS para distribuir e registar as chaves publicas para o receptor.
- O emissor usa XML-ENC para cifrar o corpo da mensagem
- Depois assina a mensagem com o XML-SIG.
- Depois de receber a mensagem, o receptor, verifica a assinatura da mensagem (possivelmente usando a chave do XKMS).
- Depois de confirmada a validade da assinatura usa o XML-ENC para decifrar o corpo da mensagem.

Posteriormente o W3C propôs o SOAP-DSIG (*SOAP Digital Signature*) como uma forma *standard* de incorporar a extensão XML-SIG directamente na especificação SOAP. O SOAP-DSIG especifica a sintaxe e as regras de processamento do elemento `SOAP-SEC`, do envelope da mensagem SOAP, que incorpora informação relativa à assinatura digital.

### 2.3.5 Ambientes de desenvolvimento

A implementação do serviço Web e a sua disponibilização requerem o uso de tecnologias, que fazem uso da definição WSDL. Neste projecto, tal como foi definido no documento WSDL, a comunicação é feita segundo o protocolo SOAP. Isto não limita a linguagem de implementação, contudo, existem algumas como o Java, o Perl e o .Net que são as mais utilizadas para estes fins.

A grande maioria das ferramentas, para trabalhar com documentos baseados em XML, são implementadas nestas linguagens. Este facto facilita a implementação do servido, já que existem muitas ferramentas de utensílio para esse trabalho.

Um serviço Web pode ser codificado de raiz, processando directamente os dados em XML. Isto é muito raro acontecer, já que existem ferramentas que fazem uma parte do trabalho. Algumas ferramentas permitem criar um esqueleto[6] do serviço, numa determinada linguagem, ficando o corpo do serviço por implementar. Algumas ferramentas também dispõem de várias funcionalidades (classes de procedimentos) que podem ser usadas pelos programadores.

[6]O esqueleto do serviço é a estrutura do código do serviço. Por exemplo, em Java, o esqueleto é a definição da estrutura das classes.

Em seguida são apresentadas duas aplicações cliente que comunicam SOAP. São implementadas em linguagens diferentes, mas comunicam com o mesmo serviço. A primeira é apresentada em 2.24 está implementada em Java, segundo a ferramenta *Axis*, enquanto que a segunda é apresentada em 2.25 está implementada em Perl, segundo a ferramenta *SOAP::Lite*.

```
1  package samples.tese;
2
3  import org.apache.axis.AxisFault; import
4  org.apache.axis.client.Call; import
5  org.apache.axis.client.Service; import
6  org.apache.axis.utils.Options; import org.apache.axis.message.*;
7  import java.io.ByteArrayInputStream; import org.w3c.dom.*; import
8  java.sql.*; import java.text.*; import javax.xml.parsers.*; import
9  javax.xml.rpc.ParameterMode; import javax.xml.rpc.namespace.QName;
10 import java.util.*;
11
12 public class Client {
13     public static void main(String [] args) throws Exception
14     {
15         Options options = new Options(args);
16         Request request = new Request();
17         Item it = new Item();
18         String source = "FCUP";
19         String ID = "rq1";
20         java.util.Date date = new java.util.Date();
21         SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");
22         String timestamp = df.format(date);
23
24         it.setObject("Aluno");
25         it.setQuery("/Aluno[ncd='960307050']");
26         it.setItemID("item1");
27         it.setUpdateDate("20020611120842");
28
29         request.setItem(new Item[] { it });
30         request.setTimestamp(timestamp);
31         request.setSource(source);
32         request.setID(ID);
33
34         Service  service = new Service();
35         Call     call    = (Call) service.createCall();
36         QName qn =  new QName("DataExchange", "queryData");
37         QName qn2 =  new QName("http://www.fc.up.pt/Meta", "Request");
38         QName qn4 =  new QName("http://www.fc.up.pt/Meta", "Item");
39         call.registerTypeMapping(Request.class, qn2,
40              new org.apache.axis.encoding.ser.BeanSerializerFactory(Request.class
                 , qn2),
41              new org.apache.axis.encoding.ser.BeanDeserializerFactory(Request.
                 class, qn2));
42
43         call.registerTypeMapping(Item.class, qn4,
44              new org.apache.axis.encoding.ser.BeanSerializerFactory(Item.class,
                 qn4),
45              new org.apache.axis.encoding.ser.BeanDeserializerFactory(Item.class,
                 qn4));
46         try {
47            call.setTargetEndpointAddress( new java.net.URL("http://depa.fc.up.pt
                 :8080/axis/servlet/AxisServlet"));
48            call.setOperationName( qn );
49            org.w3c.dom.Element temp = (org.w3c.dom.Element)  call.invoke( new
                 Object[] { request } );
50         } catch (AxisFault fault) {
51            System.out.println("Error : " + fault.toString());
52         }
53     }
```

```
54 }
```

Programa 2.24: Aplicação de um cliente SOAP em Java.

```
1  #!/usr/bin/perl
2
3  use SOAP::Lite; use XML::Parser::Lite;
4
5  my $request = SOAP::Header->name("meta:Request")->attr({"object"
6  =>
7                  "Aluno","query" => "/Aluno","sinceDate" => "20020611120842",
8                  "timestamp" => "20020611120842", "source" => "FCUP",
9                  "ID" => "123"});
10
11 my $service = SOAP::Lite -> uri("DataExchange") ->
12 service("file:protocolo.wsdl"); my $result = $service ->
13 queryData($request);
14
15 print $result;
```

Programa 2.25: Aplicação de um cliente SOAP em Perl.

A implementação de aplicações SOAP difere muito consoante as linguagens, como é visível nestes exemplos. O segundo exemplo, a aplicação implementada em Perl, é muito mais reduzida do que a implementação em Java. A escolha da linguagem mais indicada para implementar uma aplicação SOAP, depende muito da dimensão e objectivos da mesma, e do conhecimento que o programador tem das linguagens. Isto é, normalmente, um programador opta por implementar aplicações nas linguagens que está mais familiarizado.

Neste âmbito, um serviço é uma aplicação que tem como função responder a,pedidos de informação que lhe são solicitados pelas aplicações cliente de outros organismos. Como a comunicação das mensagens SOAP é feita segundo um protocolo de transporte, neste caso o HTTP, é necessário que o serviço seja disponibilizado por um servidor HTTP. Por seu lado, o servidor tem de saber qual o programa que corresponde cada serviço, para que quando lhe for feito um pedido de um serviço, o programa correspondente possa ser executado tendo a mensagem SOAP como parâmetro.

# Capítulo 3

# Metodologia de partilha de dados

Neste capítulo é abordada a questão central desta dissertação - a definição duma metodologia de partilha de dados entre sistemas heterogéneos, assente em *standards* de modelação, definição e comunicação de dados.

A metodologia proposta divide-se em três etapas: a primeira etapa aborda a modelação de dados de uma organização, segundo diagramas de classes da linguagem UML; a segunda etapa tem como objectivo a obtenção de estruturas de dados relativas ao modelo criado na etapa anterior; a última etapa apresenta a forma de comunicação dos dados através de serviços de dados. Nesta etapa também são descritas soluções de segurança que foram testadas no exemplo. Estas três etapas são apresentadas nas próximas três secções respectivamente.

## 3.1 Modelação de dados

O UML foi a linguagem escolhida para a etapa de modelação de dados. Esta secção começa por realçar alguns das características que levaram à escolha do UML. Em seguida aborda-se o tema *forward/reverse engineering* no âmbito deste projecto e apresentam-se as componentes UML que são utilizadas na modelação do sistema. Tirando partido do facto do UML ser uma linguagem extensível (vêr secção 2.1), adicionaram-se algumas funcionalidades com utilidade para a nossa metodologia que serão descritas em 3.1.1.

### 3.1.1 Utilização do UML

O UML é uma linguagem de modelação poderosa e flexível, com capacidade de descrever de aspectos estáticos, dinâmicos e ambientais dos sistemas de software.

Para além do UML ser a linguagem *standard* de modelação orientada a objectos, existem outras razões que levaram à escolha da mesma. Uma dessas razões é o facto do UML ser uma linguagem gráfica. A linguagem permite que os modelos sejam representados graficamente melhorando a percepção das relações entre as várias componentes do modelo. As diversas visões que a linguagem permite, é outra razão para a sua escolha. O mesmo sistema pode ser visto de diversas formas, segundo vários tipos de diagramas. A existência de uma forma de representação da informação UML em XML, segundo o

*standard* XMI apresentado, é uma mais valia para definir o trajecto total das etapas deste projecto, de uma forma relativamente simples. A última grande razão para a sua escolha é a existência de diversas ferramentas, para todas as plataformas, permitindo criar modelos UML sem quaisquer restrições ao nível tecnológico.

### *Forward/Reverse engineering*

Na apresentação do UML, na secção 2.1, referiu-se que esta linguagem de modelação tem uma relação muito próxima com algumas linguagens de programação. Esta relação deve-se ao facto de ser possível mapear modelos UML para essas linguagens de programação (*forward engineering*) e o processo inverso (*reverse engineering*).

Este projecto tira partido da primeira característica de forma indirecta. O usual é considerar *forward engineering* como o mapeamento de um modelo para uma linguagem de programação. No projecto esta noção é usada na representação dos modelos UML no formato XMI, que mais tarde são usados num processo de mapeamento originando definições de dados segundo *Schemas*. Por sua vez, a noção de *reverse engineering* está também presente neste projecto, também de uma forma indirecta, isto é, como uma forma de representação de sistemas já implementados e não como um processo de mapeamento entre linguagens de programação e modelos UML. Como o âmbito do projecto é criar condições para que os organismos de uma organização comuniquem, independentemente do sistema implementado ou da plataforma, tem de existir uma forma de definição da estrutura de informação da organização. Isto obriga à existência de um processo de re-engenharia para reunir as estruturas de informação existentes, segundo os vários organismos, e obter um modelo de dados que reúna a informação relevante do sistema global.

### Componentes UML utilizadas

Este projecto centrou-se apenas nos modelos de informação estática. O objectivo é criar modelos que representem a estrutura de informação da organização. Restringindo, ainda mais, o uso do UML, apenas são usados diagramas de classes. A utilização de apenas este tipo de diagrama é o suficiente para atingir os objectivos propostos.

Do conjunto de componentes dos diagramas de classes, apenas são consideradas são as seguintes:

- **Classes**. As classes são definidas como um nome. Este pode ser simples (por exemplo, Aluno) ou ter como prefixo o nome do organismo responsável pela classe (por exemplo, Secretaria::Aluno).

  Uma classe pode ter definidos um conjunto de atributos. Cada atributo pode ter definido um tipo de dados. O tipo de dados pode ser um tipo definido, ou então uma referência a uma classe que defina um novo tipo de dados.

  As classes podem ainda ser refinadas com sendo folhas ou raízes. A definição das propriedades destas classes deve ser feita apenas no pacote a que pertencem. As segundas são para indicar que uma classe é filha, hierarquicamente, de outra. As últimas indicam o inverso, uma classe é pai de outra(s).

  As classes representadas num pacote que não o delas, devem ter associado o estereotipo

`undefined` quando não estiverem estar definidas nesse pacote. Isto acontece quando existem definidas pelo menos duas classes com o mesmo, mas com prefixos diferentes no mesmo pacote. Nestes casos apenas uma dessas classes pode estar definida, por isso é que as outras devem ter o estereotipo associado.

- **Pacotes.** Os pacotes neste processo correspondem a organismos da organização. A divisão de um diagrama em pacotes (organismos), vem facilitar a atribuição de responsabilidades e direitos dos mesmos. A gestão dos objectos das classes definidas num pacote, é da responsabilidade do organismo nele definidos. O nome do pacote corresponde ao nome do organismos associado.

- **Associação.** É o tipo de relacionamento entre classes mais simples. Apenas define uma relação entre objectos de diferentes classes.

- **Dependências.** A utilização dos relacionamentos do tipo dependência, servem apenas para representar as permissões de partilha de informação entre pacotes (organismos). Quando um organismo tem permissões de importação de objectos de outro cria-se uma dependência do primeiro para o segundo, com o estereotipo `import`. A informação que o primeiro organismo obtém depende da informação do segundo.

- **Generalizações.** As generalizações são usadas segundo a sua forma natural. A única questão que merece realce é que, normalmente, as generalizações são usadas para definir relacionamentos entre classes com o mesmo nome mas pertencentes a pacotes (organismos) diferentes. Isto significa que existe uma classe genérica que reúne a totalidade dos objectos, enquanto que as especializações, têm apenas uma parte desse conjunto.

- **Composições/Agregações.** As composições e as agregações são dois tipos de relacionamentos que são dos mais usados entre classes do mesmo pacote. As composições criam uma relação em que os elementos não persistem ao grupo, isto é, quando o lado múltiplo da relação é destruído, consequentemente os elementos que o compõem são também destruídos (por exemplo, se destruirmos um painel também destruímos os botões que ele contem). Nas agregações isto não acontece já que os elementos continuam a existir.

### Cardinalidade

A cardinalidade dos relacionamentos deve estar definida apenas nos relacionamentos do tipo composição ou associação. Os valores permitidos são os seguintes: 0, 1, *, O..1, 1..*.

O tipo de cardinalidade muitos-para-muitos não é considerada, directamente, neste trabalho. No entanto, este tipo de relacionamentos pode ser criado através da divisão das duas classes e do relacionamento em três classes com dois relacionamentos. Desta forma é possível representar o relacionamento através de dois relacionamentos do tipo zero/um-para-muitos. A decisão que originou a não inclusão directa deste tipo de relacionamento, baseou-se na simplificação da construção das estruturas de dados a partir do documento XMI.

Em relação à cardinalidade dos atributos, apenas se pode indicar se esta é 1 ou 0. Para disso, deve-se associar o estereotipo `required` aos atributos com cardinalidade 1.

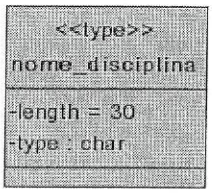



Figura 3.1: Exemplo de definição de um novo tipo de dados

### Extensões ao *standard*

Alguns dos refinamentos aos diagramas de classes apresentados podem ser consideradas extensões ao *standard*. Todavia, esses refinamentos apenas condicionam o uso de algumas componentes. As extensões introduzidas foram: a definição de novos tipos de dados e a possibilidade de definir o formato do identificador de objectos.

### Tipos de Dados

Esta extensão permite especificar, ainda mais, um diagrama de classes UML, possibilitando a definição de novos tipos de dados. Os tipos de dados podem ser definidos através de classes, com o estereotipo `type`, e não pertencem a nenhum pacote, ou seja, é uma definição de tipos de dados global.

A tabela 3.1 descreve os atributos que são usados na definição de uma classe que define um tipo de dados. Alguns necessitam que o seu valor inicial esteja preenchido, enquanto que outros necessitam apenas da associação de um tipo. A figure 3.1 apresenta um exemplo de uma classe que define um tipo de dados.

| Nome da classe | Tipo | Valor Inicial | Exemplo |
|---|---|---|---|
| **type** | tipo definido | | char |
| **length** | | tamanho | 20 |
| **pattern** | | expressão regular | [A-Z]* |
| **enumeration** | tipo | | sexo |

Tabela 3.1: Atributos de uma classe que define um tipo de dados

É também possível definir generalizações entre as classes de tipos. Em que as folhas herdam as propriedades da raiz.

### Enumerações

Associada à definição de tipos de dados, estão as enumerações, como é visível na tabela anterior. Contudo, uma classe que define um tipo de dados e que apenas indique o nome da enumeração é insuficiente. Tem de existir uma forma que indicar quais são os elementos da enumeração. Por este motivo surgiu a necessidade de criação de um novo estereotipo designado por `enumeration`. Este

estereotipo é usado na definição de uma nova classe, com o nome do tipo da enumeração. Cada atributo desta nova classe representa um elemento da enumeração.

### Identificação de objectos

Normalmente, quando se lida com objectos, tem de existir uma forma de os identificar unicamente. Esta identificação pode ser feita de diversas formas[Amb00]. Contudo, a mais usual é a adição de uma propriedade que caracterize unicamente o objecto. A definição do formato deste identificador fica em aberto neste projecto, já que existem diversas abordagens em relação à escolha da melhor forma de o fazer. Contudo, existe a possibilidade de definir o formato de dados desse atributo nos diagramas de classes neste projecto.

Caso se pretenda adicionar uma propriedade que tenha a responsabilidade de identificar unicamente um objecto, deve-se definir um tipo de dados, segundo o método e as possibilidades explicadas anteriormente, com a única restrição do nome do tipo de dados ter como sufixo a expressão `_oid`. Esta expressão serve apenas para caracterizar o tipo de dados como identificador de atributo.

### Desenho de diagramas

O processo de modelação apresentado anteriormente nada diz em relação à forma de criação dos modelos. A criação de um modelo tem de seguir um conjunto de regras para que seja criada uma estrutura de dados correcta a partir do documento XMI.

A criação do modelo de uma organização é concluído após terminado o processo de modelação. O modelo é composto por um conjunto de diagramas de classes UML, podendo existir diagramas com diferentes níveis de abstracção.

No modelo tem de existir um diagrama com o nome `global`, que é o diagrama que ilustra todos os organismos do sistema, as relações existentes entre eles e define os novos tipos de dados. Neste diagrama são representados os pacotes referentes aos organismos. Os pacotes só se podem relacionar entre si, não se pode relacionar pacotes com classes de outros pacotes e só se podem relacionar classes representadas no mesmo pacote. A definição das classes e os respectivos relacionamentos podem ser representados neste diagrama, contudo, não é necessário já que este diagrama pode ter um nível de abstracção elevado e essa definição pode ser feita apenas nos diagramas dos organismos. No entanto, esta definição fica ao critério do criador.

A definição detalhada das classes de cada organismo do diagrama global é feita criando outros diagramas de classes, com o mesmo nome dos organismos. Cada um destes diagramas vai conter a definição completa das duas componentes (classes e relacionamentos). Se estas já tiverem sido definidas no diagrama global, basta que sejam referenciadas, não sendo necessário definir classes novas.

Este passo pode ser feito de forma inversa. O diagrama global pode ser desenhado em primeiro lugar, sem classes. Depois são desenhados os diagramas dos organismos, com a definição das classes, e só depois é que as classes são referenciadas, a partir do diagrama global.

Um ponto importante são as relações entre classes de diferentes organismos. Como estas não se podem relacionar directamente, as suas definições são copiadas, adicionando o prefixo ao nome da

classe. Por exemplo, a classe A do diagrama X relaciona-se com a classe B do diagrama Y. Uma cópia da classe B do diagrama Y deve ser feita no diagrama X, ficando nesta com o nome Y:B. O mesmo teria de acontecer para definir a classe A do diagrama X no diagrama Y.

### 3.1.2 Exemplo

Nesta secção vai ser apresentado o modelo de dados do Cenário descrito na secção 1.3.3. Este modelo foi criado seguindo o processo descrito na secção anterior.

O modelo é constituído por três organismos: Departamento de Pessoal, Secretaria e o Departamento A. O Departamento de Pessoal é responsável por gerir informações relativas aos Docentes. A Secretaria é responsável por gerar toda a informação referente aos Alunos, Disciplinas e Inscrições. Como cada Disciplina tem associado um Docente[1], este organismo necessita de informações do Departamento de Pessoal referentes aos Docentes. Como as notas dos Exames são geradas pelos departamentos, a Secretaria necessita de as obter dos mesmos[2]. O Departamento A necessita de informações das Disciplinas, Alunos, inscrições e Exames para poder trabalhar e também gerar as notas dos Exames, que são mais tarde pedidas pela Secretaria.

#### Departamento de Pessoal

No Departamento de Pessoal apenas se define a classe `Docente`, com os atributos relativos ao código (`cod`) e ao `nome`. O atributo `cod` é um atributo obrigatório (tem associado o estereotipo *required*). O atributo `nome` é do tipo `nome_pessoa` que é apresentado mais à frente. A figura 3.2 apresenta a definição do modelo de dados do Departamento de Pessoal.

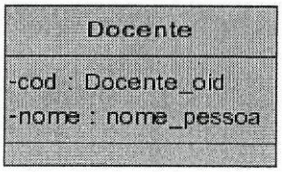



Figura 3.2: Definição do organismo Departamento de Pessoal.

#### Secretaria

A Secretaria define as classes `Aluno`, `Inscrição`, `Disciplina` e `Exame`. A classe `Aluno` tem definidos os atributos `ncd` e `nome`. O primeiro tem associado o tipo de dados `Aluno_oid`. A classe `Disciplina` tem definidos o atributo `cad` tendo associado o tipo de dados `Disciplina_oid` e o atributo nome que tem associado o tipo de dados `nome_disciplina`. A classe `Inscricao` tem definido o atributo `data` do tipo `date`. A classe `Exame` tem definido o atributo `data_exame`, que é do tipo `date`, e o atributo `nota` do tipo `nota`.

[1]Para simplificar assumiu-se que a uma Disciplina apenas está associado um Docente.

[2]Neste exemplo é apenas apresentado o Departamento A, contudo, poderiam ter sido adicionados mais.

A associação entre as classes `Aluno` e `Inscricao` é do tipo composição já que uma Inscrição tem de ter associado um Aluno e um Aluno pode ter mais do que uma Inscrição. O mesmo acontece com a associação entre as classes `Disciplina` e `Inscricao`, porque uma Inscrição tem de ser referente a uma Disciplina e esta por ter associadas várias inscrições. A associação entre as classes `Exame` e `Inscricao` também é do tipo composição já que um Exame é referente a uma Inscrição e esta pode ter associados vários Exames.

A associação entre a classe `DepPessoal::Docente` e a classe `Disciplina` também é uma composição porque uma Disciplina tem de ter sempre associado um regente. Um Docente pode reger várias Disciplinas. A associação entre a classe `Exame` e a classe `DepA::Exame` é uma generalização porque a informação provêm dos organismos que especializam a classe. A figura 3.3 representa o modelo de dados do organismo Secretaria.

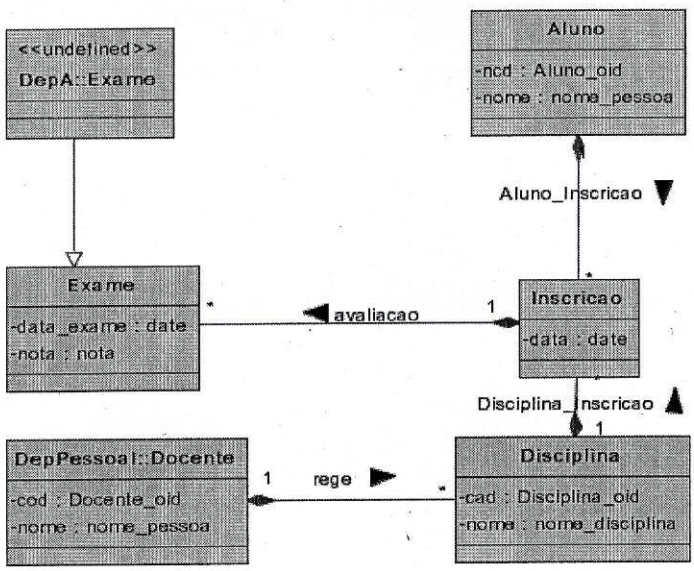



Figura 3.3: Definição do organismo Secretaria.

### Departamento A

O organismo Departamento A importa do organismo Secretaria as classes: `Aluno`, `Inscricao`, `Disciplina`, `Docente` e `Exame`. A figura 3.4 representa o modelo do Departamento A.

### Diagrama Global

No diagrama global são representados os pacotes referentes aos organismos. São definidas relações de importação entre os pacotes. As classes associadas aos pacotes podem estar representadas neste diagrama. A figura 3.5 representa este diagrama.

Os tipos de dados que não pertencem à especificação UML devem ser definidos neste diagrama como classes. Neste diagrama foram definidos os tipos de dados: `nota`, `nome_pessoa`, `nome_disciplina`, `oid`, `Aluno_oid` e `Disciplina_oid`. Os últimos dois definem propriedades que identificam unicamente

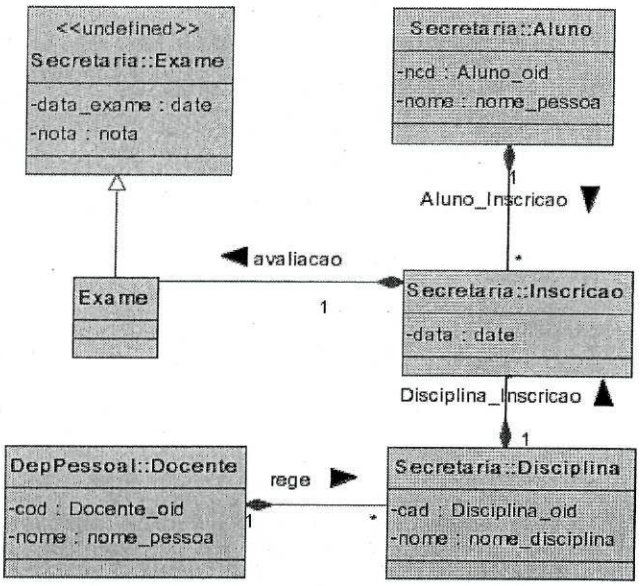



Figura 3.4: Definição do organismo Departamento A.

os objectos, são também propriedades obrigatórias.

## 3.2 Estruturas de Dados

Como resultado do processo de modelação proposto na secção 3.1, pretende-se obter uma definição de dados num formato genérico, independente da plataforma e do sistema. O formato de dados com essas características que foi escolhido é o XML, mais concretamente a especificação *XML Schema*, apresentada na secção 2.2.4. Para obter uma definição de dados neste formato, a partir dos modelos de dados UML propostos, tem de existir um processo de mapeamento.

Como foi descrito na secção 2.2.6, existe uma especificação XML - o XMI - usada para trocar informação entre aplicações num formato genérico.

O processo de mapeamento é feito definindo um conjunto de regras de transformação num folha de estilos XSLT. Este processo torna-se automático, já que o documento XSLT é aplicado sobre o documento XMI, por um *parser* XSLT, gerando *Schemas*. O número de documentos resultantes equivale ao número de organismos do modelo de dados. A figura 3.6 demonstra este processo de mapeamento.

Este processo permite um desenvolvimento rápido de definições de dados noutras linguagens que não os *Schemas*, como por exemplo DTD's. Permite também fazer alterações ao modelo com custos reduzidos. Para que as alterações ao modelo de dados UML se reflictam na definição de dados, basta aplicar a folha de estilos XSLT novamente sobre o documento XMI.

A secção 3.2.2 apresenta as regras de mapeamento do modelo de dados UML, no formato XMI, para os *Schemas*. Também descreve a estrutura dos *Schemas* resultantes, que foi considerada. A secção 3.2.2 descreve o documento XSLT, ou seja, descreve a sequência de transformações do documento.

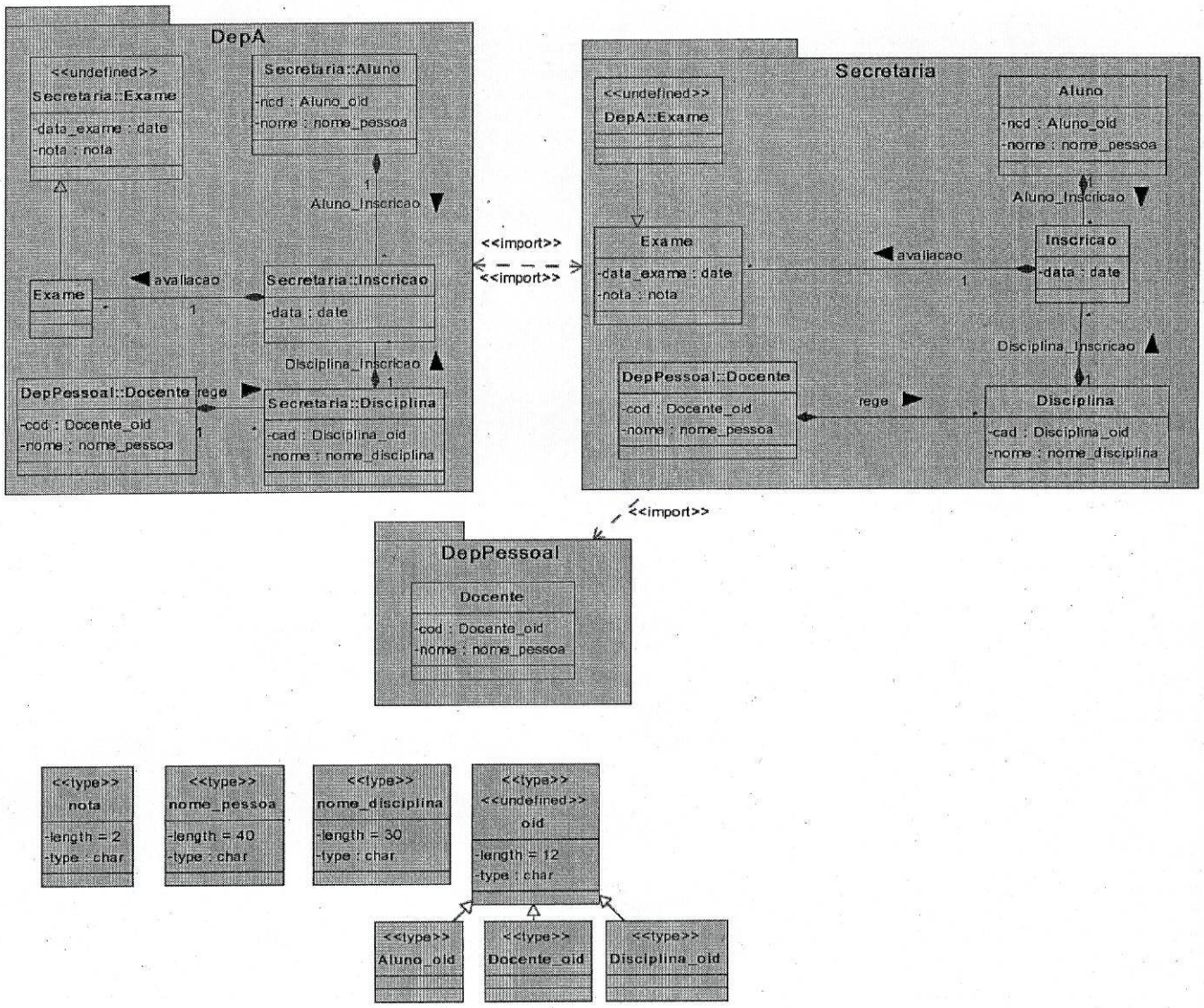



Figura 3.5: Definição do sistemas global.

Por fim, na secção 3.2.3 são apresentados dois exemplos de outros formatos - SQL e ODL - do modelo de dados, obtidos a partir dos *Schemas* obtidos.

### 3.2.1 Definição

A componente fundamental do processo de mapeamento entre os modelos UML e os *Schemas* é a folha de estilos XSLT. A sua criação teve algumas considerações. A primeira é o facto de se pretender definir apenas uma folha de estilos para qualquer modelo que se crie, o que obriga a que esta seja totalmente independente do modelo. A segunda é a necessidade de reflectir a informação dos modelos nos documentos resultantes. A terceira é garantir que os documentos resultantes sejam válidos.

As regras de mapeamento têm como ponto de partida o modelo de dados no formato XMI, originando um conjunto de elementos XML que definem um documento *XML Schema* válido. A tabela 3.2

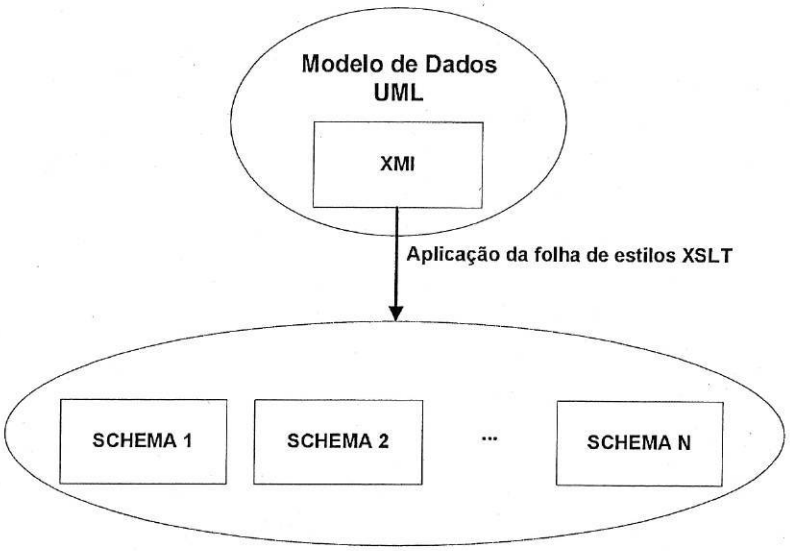



Figura 3.6: Processo de Mapeamento.

apresenta essas regras de mapeamento, em que só são consideradas as componentes UML indicadas na secção 3.1.1 do processo de modelação proposto.

Estas regras apenas indicam, de forma simples, que componentes do modelos de dados fazem parte do domínio das regras e quais as definições de dados que resultam da aplicação das regras. É necessário ter uma noção de como estão organizadas as definições de dados, no contexto do documento, para se perceber melhor a forma como elas se relacionam. Essa organização é apresentada a seguir.

Os *XML Schemas* resultantes são documentos XML com um estrutura derivada das etapas do processo de mapeamento. Desta forma simplificou-se o processo de mapeamento e estruturou-se os documentos resultantes para melhorar a sua compreensão.

A estrutura destes documentos está dividida em três partes:

- Definição de elemento que incorpora objectos
- Definição dos tipos complexos relativos às classes
- Definição de elementos relativos aos tipos de dados

Como as componentes do modelo de dados UML apenas geram tipos complexos, as instâncias dum esquema, obtido por simples mapeamento dum documento XMI, não poderia conter elementos. Tem de ser fornecida uma definição de elementos, que usem os tipos complexos definidos, para que possam existir objectos das classes nas instâncias do esquema. A solução foi definir um elemento, como o nome `SetDataObject`, que pode conter um conjunto qualquer de elementos correspondentes a objectos do modelo. Estes elementos referem tipos complexos com o mesmo nome que definem os elementos.

Depois são definidos os tipos complexos referentes às classes do modelo de dados, que incorporam elementos correspondentes aos atributos. Se uma classe não tiver definido uma propriedade identificadora, é criado um atributo, em que o nome é o mesmo da classe seguido de `_oid`, que assume esse papel.

| UML(estereotipo) | XML Schema | Observações |
|---|---|---|
| **Classe** | **Tipo complexo.** | |
| **Classe**(type) | **Tipo complexo** | Novo tipo de dados que é usado por outros elementos. Contem outros elementos como propriedades restritivas. |
| **Classe**(enumeration) | **Os elementos da enumeração** | Elementos associados ao respectivo tipo de dados. |
| **Atributo** | **Elemento** | É definido o tipo de dados e cardinalidade. Fica associado ao tipo complexo da respectiva classe. |
| **Atributo**(required) | **Elemento.** | A única diferença para o anterior é que este é obrigatório. |
| **Associação** | **Elemento.** | É criado na classe que tem participação maior do que 1. Se for uma associação do tipo um-para-um, o elemento é criado na primeira extremidade da associação (esta extremidade é o ponto de partida na criação da associação). |
| **Agregação/Composição** | **Elemento.** | As agregações ou composições são tipos particulares de associações que obrigam à existência de uma instância da classe que é referenciada. |
| **Generalização** | **Elemento.** | Quando existe uma generalização de classes, apenas as classes filho é que são consideradas, herdando dos pais os atributos e possíveis referências a outras classes. |

Tabela 3.2: Regras de Mapeamento

O documento 3.1 apresenta um excerto de um *Schema* e o documento 3.2 é uma instância desse *Schema*. Nesta figura é visível que a existência dos elementos `Aluno` e `Disciplina` deve-se ao facto de fazerem parte do elemento `SetDataObject`, já que o processo de mapeamento apenas cria tipos complexos e elementos incorporados e não elementos de topo.

```
1  <?xml version="1.0" encoding="utf-8"?> <xsd:schema
2  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3  xmlns:data="http://www.dataexchangeservice.com/Data-types"
4  elementFormDefault="qualified" attributeFormDefault="unqualified"
5  targetNamespace="http://www.dataexchangeservice.com/Data-types">
6
7     <xsd:element name="SetDataObject" type="data:SetDataObject"/>
8     <xsd:complexType name="SetDataObject">
9        <xsd:choice maxOccurs="unbounded">
10          <xsd:element name="Disciplina" type="data:Disciplina"
11                       maxOccurs="unbounded"/>
```

```
12            <xsd:element name="Aluno" type="data:Aluno"
13                         maxOccurs="unbounded"/>
14            <xsd:element name="Exame" type="data:Exame"
15                         maxOccurs="unbounded"/>
16            <xsd:element name="Inscricao" type="data:Inscricao"
17                         maxOccurs="unbounded"/>
18            <xsd:element name="Docente" type="data:Docente"
19                         maxOccurs="unbounded"/>
20        </xsd:choice>
21    </xsd:complexType>
22
23    <xsd:complexType name="Disciplina">
24        <xsd:all>
25            <xsd:element name="cad" type="data:Disciplina_oid"
26                         minOccurs="0" maxOccurs="1"/>
27            <xsd:element name="nome" type="data:nome_disciplina"
28                         minOccurs="0" maxOccurs="1"/>
29        </xsd:all>
30    </xsd:complexType>
31    <xsd:complexType name="Aluno">
32        <xsd:all>
33            <xsd:element name="ncd" type="data:Aluno_oid"
34                         minOccurs="0" maxOccurs="1"/>
35            <xsd:element name="nome" type="data:nome_pessoa"
36                         minOccurs="0" maxOccurs="1"/>
37        </xsd:all>
38    </xsd:complexType>
39
40  ......
```

Programa 3.1: Excerto de um *Schema*.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <SetDataObject>
3     <Aluno>
4        <ncd>960307001</ncd>
5        <nome>Manuel António</nome>
6     </Aluno>
7     <Disciplina>
8        <cad>1996071101</cad>
9        <nome>Programação de Computadores</nome>
10    </Disciplina>
11 </SetDataObject>
```

Programa 3.2: Instância do *Schema* 3.1.

Na última parte do documento, estão definidos os tipos de dados definidos no modelo de dados. Estas definições, de tipos de dados simples, incorporam elementos que definem restrições. São também definidos, por defeito, os tipos de dados dos elementos identificadores das classes que não tenham sido definidos no modelo de dados.

### 3.2.2 Mapeamento

Como foi referido anteriormente, os documentos resultantes, do processo de mapeamento, têm uma estrutura. Esta é caracterizada por três partes distintas: o elemento que contem as instâncias dos objectos do modelo de dados, os tipos complexos que definem as classes do modelo e os tipos simples relativos aos novos tipos de dados. Para obter uma estrutura com estas características, a folha de estilos XSLT, deve seguir um conjunto de passos.

- **Ponto de partida.** Para se começar a processar o documento, é necessário definir um ponto de partida. Esse ponto é sempre definido através de um *template*. O primeiro *template* encaixa no início do documento.

  Como foi referido, o processo de mapeamento pode dar resultado a um ou mais documentos (*Schemas*). Esse número equivale ao número de organismos representados no modelo de dados, já que vai existir um documento por organismo. A primeira coisa que a folha de estilos faz é determinar os pacotes existentes, e para cada um deles definir a respectiva estrutura de dados.

- **Documento por organismo.** A definição da estrutura de um organismo é feita em vários passos. O primeiro é criar um elemento, o `SetDataObject`, que contem a definição dos elementos relativos às instâncias das classes do modelo de dados. Apenas são definidos os elementos que dizem respeito ao organismo. A estrutura destes elementos é definida mais tarde como tipos complexos, o que obriga à existência de referências aos tipos complexos com o mesmo nome dos elementos.

- **Tipos complexos por classe.** O passo seguinte é a definição de um tipo complexo por cada classe dum organismo. Cada um incorpora elementos relativos às propriedades da classe associada.

  É verificada a existência de uma propriedade que identifique unicamente as instâncias da classe, isto é, uma propriedade que tenha como sufixo `_oid`. Caso não esteja definido, é criado um elemento com o seu nome e o sufixo `_oid`. É também criada uma referência a um tipo de dados que vai ser definido mais tarde.

  Em seguida são criados os restantes elementos relativos às propriedades da classe, identificando o seu tipo de dados ou a referência à sua definição. É também definida a cardinalidade da propriedade, que pode ser 1 ou 0 (1 indica que o elemento é obrigatório enquanto que 0 indica que não), dependendo da propriedade, no modelo de dados, ter associado o estereotipo `required`.

  Depois são definidos os atributos que resultam de generalizações, ou seja, as classes filho têm de herdar atributos das classes pai.

  Por último são criados os atributos que resultam de relacionamentos do tipo associação, agregação ou composição. Neste casos é criado um atributo na classe que tem uma participação, no relacionamento, maior do que 1. Esse atributo tem o nome da outra classe com o sufixo `_oid`, para identificar unicamente a instância da outra classe. A cardinalidade do atributo depende do tipo de relacionamento, já que uns obrigam à existência de uma referência, outros não. No caso de ser uma associação do tipo um-para-um, o elemento é criado na primeira extremidade da associação (esta extremidade é o ponto de partida na criação da associação). No secção 3.1 foi referido que não são considerados relacionamentos do tipo muitos-para-muitos porque estes podem-se decompor em dois do tipo um-para-muitos. No entanto, se estes forem representados no modelo de dados, o resultado do mapeamento é o mesmo das associações um-para-um.

- **Tipos de dados identificadores.** O passo seguinte é definir os tipos de dados dos elementos referentes às propriedades identificadoras, que não foram definidas no modelo de dados, mas que fazem parte desta estrutura de dados. Foi assumido, por omissão, o tipo `string` com o tamanho 20, para estas definições. O excerto 3.3 exemplifica duas definições de tipos identificadores.

```
1    <xsd:simpleType name="Exame_oid">
2       <xsd:restriction base="xsd:string">
3          <xsd:maxLength value="20"/>
```

```
4         </xsd:restriction>
5      </xsd:simpleType>
6
7      <xsd:simpleType name="Secretaria::Inscricao_oid">
8         <xsd:restriction base="xsd:string">
9            <xsd:maxLength value="20"/>
10        </xsd:restriction>
11     </xsd:simpleType>
```

Programa 3.3: Definição de tipos identificadores.

- **Tipos de dados.** Por fim são criados todos os tipos de dados que foram definidos no modelo de dados (classes que têm associado o estereotipo `type`). Estes tipos complexos podem definir o tipo de dados, o tamanho máximo, uma expressão regular ou de elementos de uma enumeração. São apresentados alguns exemplos de definições de tipos de dados em 3.4.

```
1      <xsd:simpleType name="Exame_oid">
2         <xsd:restriction base="xsd:string">
3            <xsd:maxLength value="20"/>
4         </xsd:restriction>
5      </xsd:simpleType>
6
7      <xsd:simpleType name="Secretaria::Inscricao_oid">
8         <xsd:restriction base="xsd:string">
9            <xsd:maxLength value="20"/>
10        </xsd:restriction>
11     </xsd:simpleType>
```

Programa 3.4: Definição de tipos de dados definidos no modelo de dados.

### 3.2.3 Criação

Os *Schemas* obtidos através do processo de mapeamento de modelos de dados UML, no formato XMI, são uma nova estrutura e um novo formato de informação independente do organismo. Estes documentos podem ser usados para diversos fins:

- Formato de dados para comunicação entre organismos. Este é o principal fim para que eles foram criado nesta metodologia.
- Criar novos formatos de visualização de dados. Aplicando novamente uma folha de estilos XSLT sobre estes documentos podem-se obter outros formatos de dados que facilitam a visualização da informação, como por exemplo HTML.
- Criar outros estruturas de dados. Aplicando novamente uma folha de estilos XSLT sobre estes documentos podem-se obter outras estruturas de dados, nomeadamente estruturas específicas aos sistemas de armazenamento de informação como o SQL (*Structured Query Language*).

Foi referido na abordagem ao problema na secção 1.3.2 que a possibilidade de gerar outras estruturas de dados a partir dos *Schemas* poderia ser bastante útil. Isto torna-se vantajoso quando se pretende criar uma estrutura de dados específica a um sistema de armazenamento de dados de um novo organismo. Para este caso poderia-se usar um definição de dados gerada automaticamente. Suponhamos que esse novo sistema vai ser implementado sobre uma base de dados relacional, em que a linguagem utilizada é o SQL. É possível criar a estrutura de dados SQL a partir do *Schema*. Para isso basta

fazer um novo mapeamento, em que o documento fonte é um *Schema* e o resultado é um documento SQL.

Para demonstrar a possibilidade de obtenção de esquemas noutras linguagens a partir dos esquemas obtido, foram criados mais dois documentos de transformação XSLT, que podem ser aplicados sobre os *Schemas* e que geram novos esquemas em SQL e ODL (*Object Definition Language*).

### SQL

O SQL é um *standard* ANSI (*American National Standards Institute*) para trabalhar com sistemas de bases de dados. É usado pela grande maioria dos fabricantes de sistemas de gestão de bases de dados. As instruções SQL permitem não só criar e alterar a estrutura de uma base de dados, como também alterar e consultar informação nas bases de dados[ANSI].

As bases de dados contêm objectos designados por tabelas. Essas tabelas são compostas por colunas que são os campos (propriedades) da tabela. As linhas das tabelas são os registos (instâncias). Comparando com um sistema orientado a objectos, uma tabela é uma classe, os seus atributos são as propriedades e as instâncias são os registos. Cada tabela tem definida uma chave, que pode ser um grupo qualquer de campos. Esta serve para identificar unicamente cada registo. Se a chave for composta por apenas um campo, esta tem um papel idêntico ao do identificador nas classes dos sistemas orientados a objectos.

O SQL contem um conjunto de instruções de criação e alteração da estrutura da base de dados, contudo, o processo de criação da base de dados pode ser feito com apenas duas dessas instruções:

- `DROP TABLE`. Esta instrução permite eliminar uma tabela. Esta instrução é opcional, mas a sua utilização garante que uma tabela não existe no momento da sua criação.
- `CREATE TABLE`. É a instrução que permite criar uma nova tabela.

O código 3.5 apresenta um exemplo que ilustra a criação de uma tabela com o nome `Aluno` através das instruções descritas.

```
1 DROP TABLE Aluno;
2 CREATE TABLE Aluno (
3         ncd                 varchar(20)  NOT NULL,
4         nome                varchar(20),
5         PRIMARY KEY (ncd)
6 );
```

Programa 3.5: Definição de uma tabela em SQL.

### ODL

Os principais concorrentes dos SGBD's (Sistemas de Gestão de Bases de Dados) Relacionais, são os OODBMS (*Object Oriented Database Management Systems*). São sistemas orientados a objectos que têm funcionalidades idênticas aos sistemas relacionais, como por exemplo a linguagem de interrogação OQL (*Object Query Language*). Juntamente com estes sistemas surgiu uma linguagem de definição de dados, a ODL (*Object Data Language*)[ODL].

O ODL é uma linguagem de definição de dados orientada a objectos, que permite definir interfaces[3] de classes de objectos, as suas propriedades e os relacionamentos entre elas. É também possível definir os tipos de dados das propriedades e a chave do interface.

O código 3.6 apresentada um exemplo de uma definição de um interface em ODL.

```
1  interface Aluno
2      (key (ncd))
3  {
4          attribute  string ncd;
5          attribute  string nome;
6  };
```

Programa 3.6: Definição de um interface em ODL.

### Regras de mapeamento

A obtenção de novos esquemas de dados é feita aplicando documentos de transformação XSLT aos *Schemas* obtidos. As regras definas nesses documentos são descritas em seguida e tiram partido da forma estruturada de como os *Schemas* são gerados.

Para poder criar uma estrutura de uma base de dados em SQL é necessário um conjunto de instruções, que podem ser obtidos automaticamente a partir dos *Schemas* resultantes. A questão fundamental neste processo é criar uma tabela por cada tipo complexo, que represente uma classe no modelo de dados. Em primeiro lugar é criada uma instrução `DROP TABLE`. Esta instrução é opcional, já que só serve para garantir que não existe nenhuma tabela com o mesmo nome na base de dados, no momento da criação. Para criar esta instrução é necessário determinar o nome, que vai corresponder ao nome da tabela.

Em seguida é necessário construir uma instrução `CREATE TABLE` para definir a estrutura da tabela. Esta instrução define o nome da tabela, que é o mesmo da classe, e um campo por cada elemento do tipo complexo. A definição dos tipos de dados dos campos é feita com algumas limitações, porque os tipos de dados aceites pelo SQL não são os mesmo dos *Schemas*. Foi feito um mapeamento entre alguns tipos de dados dos *Schemas* para tipos de dados SQL[4]. As enumerações são mapeadas para uma restrição *CHECK*. Quanto à cardinalidade dos campos, esta é definida como *NOT NULL* nos atributos obrigatórios. Em seguida é adicionada a caracterização da chave da tabela, que corresponde ao atributo identificador da classe. Por fim são adicionadas as chaves externas resultantes dos relacionamentos entre as classes.

Para criar uma definição de dados orientada a objectos (ODL), é necessário definir as interfaces das classes. Estas são usadas na implementação das classes por parte dos OODBMS. A cada tipo complexo deve ser criada uma interface correspondente a uma classe do modelo de dados. Em primeiro lugar é necessário definir o nome da interface, que é o mesmo do tipo complexo. Depois é definida a chave da interface que corresponde ao atributo identificador da classe. Em seguida são definidos os atributos do interface que são os mesmos do tipo complexo. Em relação aos tipos de dados dos atributos, acontece o mesmo que no mapeamento para SQL. O mapeamento entre os tipos de dados dos *Schemas* para os do ODL é limitado, mas o processo de expansão é simples. As

[3]Um interface é uma definição de uma classe, não uma implementação. As classes é que implementam interfaces.
[4]O mapeamento pode ser facilmente alargado a um novo tipo de dados com apenas três linhas de código XSLT.

enumerações são definidas directamente. A cardinalidade não é definida. Quanto às referências a outras classes são também definidas como uma propriedade.

### 3.2.4 Exemplo

Nesta secção são apresentados os resultados do processo de mapeamento do modelo de dados UML da secção 3.1.2. Vão ser apresentados só os documentos referentes ao organismo Departamento A.

#### Schema

O *Schema* completo relativo a este organismo pode ser visto no documento 3.7. Analisando o documento por partes, é visível um elemento inicial o `SetDataObject` que define os elementos referentes ao objectos do modelo de dados. É feita uma referência aos tipos complexos que definem esses elemento.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsd:schema
3  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4  xmlns:data="http://www.dataexchangeservice.com/Data-types"
5  elementFormDefault="qualified" attributeFormDefault="unqualified"
6  targetNamespace="http://www.dataexchangeservice.com/Data-types">
7     <xsd:element name="SetDataObject" type="data:SetDataObject"/>
8     <xsd:complexType name="SetDataObject">
9        <xsd:choice maxOccurs="unbounded">
10          <xsd:element name="Exame" type="data:Exame"
11                       maxOccurs="unbounded"/>
12          <xsd:element name="Inscricao" type="data:Inscricao"
13                       maxOccurs="unbounded"/>
14          <xsd:element name="Disciplina" type="data:Disciplina"
15                       maxOccurs="unbounded"/>
16          <xsd:element name="Aluno" type="data:Aluno"
17                       maxOccurs="unbounded"/>
18          <xsd:element name="Docente" type="data:Docente"
19                       maxOccurs="unbounded"/>
20       </xsd:choice>
21    </xsd:complexType>
22    <xsd:complexType name="Exame">
23       <xsd:all>
24          <xsd:element name="id" type="data:Exame_oid"
25                       minOccurs="1" maxOccurs="1"/>
26          <xsd:element name="data_exame" type="xsd:date"
27                       minOccurs="0" maxOccurs="1"/>
28          <xsd:element name="nota" type="data:nota"
29                       minOccurs="0" maxOccurs="1"/>
30          <xsd:element name="Inscricao" type="data:Inscricao_oid"
31                       minOccurs="0" maxOccurs="1"/>
32       </xsd:all>
33    </xsd:complexType>
34    <xsd:complexType name="Inscricao">
35       <xsd:all>
36          <xsd:element name="id" type="data:Inscricao_oid"
37                       minOccurs="1" maxOccurs="1"/>
38          <xsd:element name="data" type="xsd:date"
39                       minOccurs="0" maxOccurs="1"/>
40          <xsd:element name="Disciplina" type="data:Disciplina_oid"
41                       minOccurs="0" maxOccurs="1"/>
42          <xsd:element name="Aluno" type="data:Aluno_oid"
43                       minOccurs="0" maxOccurs="1"/>
44       </xsd:all>
```

```
45      </xsd:complexType>
46      <xsd:complexType name="Disciplina">
47         <xsd:all>
48            <xsd:element name="cad" type="data:Disciplina_oid"
49                         minOccurs="1" maxOccurs="1"/>
50            <xsd:element name="nome" type="data:nome_disciplina"
51                         minOccurs="0" maxOccurs="1"/>
52            <xsd:element name="Docente" type="data:Docente_oid"
53                         minOccurs="0" maxOccurs="1"/>
54         </xsd:all>
55      </xsd:complexType>
56      <xsd:complexType name="Aluno">
57         <xsd:all>
58            <xsd:element name="ncd" type="data:Aluno_oid"
59                         minOccurs="1" maxOccurs="1"/>
60            <xsd:element name="nome" type="data:nome_pessoa"
61                         minOccurs="1" maxOccurs="1"/>
62         </xsd:all>
63      </xsd:complexType>
64      <xsd:complexType name="Docente">
65         <xsd:all>
66            <xsd:element name="cod" type="data:Docente_oid"
67                         minOccurs="1" maxOccurs="1"/>
68            <xsd:element name="nome" type="data:nome_pessoa"
69                         minOccurs="0" maxOccurs="1"/>
70         </xsd:all>
71      </xsd:complexType>
72      <xsd:simpleType name="Exame_oid">
73         <xsd:restriction base="xsd:string">
74            <xsd:maxLength value="20"/>
75         </xsd:restriction>
76      </xsd:simpleType>
77      <xsd:simpleType name="Inscricao_oid">
78         <xsd:restriction base="xsd:string">
79            <xsd:maxLength value="20"/>
80         </xsd:restriction>
81      </xsd:simpleType>
82      <xsd:simpleType name="nome_pessoa">
83         <xsd:restriction base="xsd:char">
84            <xsd:maxLength value="40"/>
85         </xsd:restriction>
86      </xsd:simpleType>
87      <xsd:simpleType name="nota">
88         <xsd:restriction base="xsd:char">
89            <xsd:maxLength value="2"/>
90         </xsd:restriction>
91      </xsd:simpleType>
92      <xsd:simpleType name="nome_disciplina">
93         <xsd:restriction base="xsd:char">
94            <xsd:maxLength value="30"/>
95         </xsd:restriction>
96      </xsd:simpleType>
97      <xsd:simpleType name="Docente_oid">
98         <xsd:restriction base="xsd:char">
99            <xsd:maxLength value="12"/>
100        </xsd:restriction>
101     </xsd:simpleType>
102     <xsd:simpleType name="Disciplina_oid">
103        <xsd:restriction base="xsd:char">
104           <xsd:maxLength value="12"/>
105        </xsd:restriction>
106     </xsd:simpleType>
107     <xsd:simpleType name="Aluno_oid">
108        <xsd:restriction base="xsd:char">
109           <xsd:maxLength value="12"/>
110        </xsd:restriction>
```

```
111     </xsd:simpleType>
112 </xsd:schema>
```

Programa 3.7: Schema do organismo Departamento A.

Depois define-se um conjunto de tipos complexos que determinam os elementos que fazem parte do elemento referido anteriormente. Estes tipos complexos incorporam a definição das propriedades dos elementos através de atributos. Cada tipo tem um elemento identificador.

Em seguida são visíveis alguns tipos de dados que definem os tipos de dados identificadores. A definição destes tipos é feita devido à sua omissão no modelo de dados.

Por fim são definidos os tipos de dados simples que foram definidos no modelo de dados UML. Fazem parte dessa definição propriedades relativas ao tipo de dados, tamanho, valor inicial e valores de enumeração.

### Outras estruturas de dados

Como foi referido, a partir dos *Schemas* obtidos, podem ser geradas outras estruturas de dados. Para demonstrar este facto foram geradas duas estruturas a partir do *Schema* do Departamento A.

O código 3.8 apresenta a estrutura de dados gerada em SQL, enquanto que o código 3.9 apresenta a mesma estrutura gerada em ODL.

```
1  DROP TABLE Aluno;
2  CREATE TABLE Aluno (
3          ncd                 varchar(12)  NOT NULL,
4          nome                varchar(20),
5          PRIMARY KEY (ncd)
6  );
7  DROP TABLE Disciplina;
8  CREATE TABLE Disciplina (
9          cad                 varchar(12)  NOT NULL,
10         nome                varchar(20),
11         PRIMARY KEY (cad)
12 );
13 DROP TABLE Docente;
14 CREATE TABLE Docente (
15         cod                 varchar(12),
16         nome                varchar(20),
17         PRIMARY KEY (cod)
18 );
19 DROP TABLE Inscricao;
20 CREATE TABLE Inscricao (
21         id                  varchar(20)  NOT NULL,
22         data                date,
23         Aluno               varchar(20)  NOT NULL,
24         Disciplina          varchar(20)  NOT NULL,
25         PRIMARY KEY (id),
26         FOREIGN KEY (Aluno) REFERENCES Aluno (ncd),
27         FOREIGN KEY (Disciplina) REFERENCES Disciplina (cad)
28 );
29 DROP TABLE Exame;
30 CREATE TABLE Exame (
31         id                  varchar(20)  NOT NULL,
32         data_exame          date,
33         nota                varchar(2),
34         Inscricao           varchar(20)  NOT NULL,
35         PRIMARY KEY (id),
```

```
36          FOREIGN KEY (Inscricao) REFERENCES Inscricao (id)
37  );
```

Programa 3.8: Estrutura de dados em SQL.

```
1   interface Exame
2       (key (id))
3   {
4           attribute  string id;
5           attribute  string data_exame;
6           attribute  string nota;
7           references Inscricao Exame_ref;
8   };
9   interface Disciplina
10      (key (cad))
11  {
12          attribute  string cad;
13          attribute  string nome;
14  };
15  interface Inscricao
16      (key (id))
17  {
18          attribute  string id;
19          attribute  date data;
20          references Aluno Inscricao_ref;
21          references Disciplina Inscricao_ref;
22  };
23  interface Aluno
24      (key (ncd))
25  {
26          attribute  string ncd;
27          attribute  string nome;
28  };
29  interface Docente
30      (key (cod))
31  {
32          attribute  string cod;
33          attribute  string nome;
34  };
```

Programa 3.9: Estrutura de dados em ODL.

## 3.3 Serviços de dados

Obtida a definição de dados, fica concluído o segundo objectivo deste projecto. A cada organismo fica associado o respectivo *Schema*, representando a estrutura da informação que este pode comunicar.

O que se pretende de seguida é comunicar informação entre os organismos. Para isso é necessário relembrar que um organismo é uma entidade abstracta. No âmbito do projecto, um organismo é um nó na rede de comunicação (WWW).

A comunicação pode ser feita através da simples troca de documentos entre os organismos. Através dos *Schemas* é possível validar a estrutura dos documento de informação comunicados usando um *parser* XML. Contudo, esta prática não é suficiente já que se pretende implementar um sistema de comunicação de informação.

Obtida uma definição de dados (*Schemas*) e sabendo qual é a plataforma de comunicação, falta definir o modelo de comunicação para completar o sistema de comunicação de informação. Esse modelo vai

ser apresentado nesta secção.

Como nos dias de hoje, nenhum sistema de informação é usado se não implementar algumas medidas de segurança, este sistema também o faz. Na secção 2.3.4 são apresentadas algumas considerações de segurança que podem ser implementadas em modelos de comunicação como o usado. Por fim é apresentada a solução usada no sistema.

Para que seja possível que os sistemas heterogéneos dos organismos comuniquem, é necessário um modelo independente da plataforma e que respeite as definições de dados obtidas. Outra das questões relacionadas com o modelo é o tipo de comunicação que deve ser feita. O que se pretende é que os organismos obtenham informação uns dos outros. Cada organismo tem de disponibilizar um serviço de fornecimento de informação, para que os outros possam fazer pedidos.

O modelo de comunicação usando foram os serviços Web. Como foi referido no Capítulo 2, os serviços Web baseiam-se em tecnologias XML e podem ser implementados em qualquer plataforma. O que se pretende é criar serviços Web que usem as definições de dados obtidas para definir a estrutura da informação comunicada entre os organismos.

A seguir apresenta-se a definição do serviço Web que foi criado como exemplo de um modelo de comunicação. Essa definição é feita através de um documento WSDL. Depois são apresentadas as tecnologias usadas para implementar o serviço Web, baseando-se no documento WSDL definido, segundo o protocolo de comunicação SOAP. Por fim é descrito o funcionamento do cliente e do serviço que foram criados.

### 3.3.1 Definição do serviço

Os documentos WSDL estão divididos em várias secções[5]. Normalmente, um documento WSDL tem uma estrutura com 6 secções: as definições, os tipos, as mensagens, os tipos de porta, as associações e os serviços. A ordem como são definidas as secções é arbitrária. Para se entender melhor cada uma das secções e o relacionamento existente entre elas, abordaremos cada uma delas pela ordem que, normalmente, os documento WSDL estão estruturados.

Antes se começarmos a descrever cada uma das secções vamos explicar a relação entre o documento WSDL e o funcionamento do serviço. Para isso é melhor observar a figura 3.7, que representa a comunicação existente entre um cliente e um serviço Web. Da figura é possível visualizar uma comunicação do tipo `Pedido-Resposta` que corresponde a uma operação (chamada de uma função com parâmetros e valor de retorno). Os parâmetros são enviados numa mensagem de pedido SOAP, são aceites na respectiva porta do servidor. A mensagem SOAP de resposta é enviada com o resultado da operação para o cliente. Quanto às mensagens elas são compostas por várias partes.

A figura 3.8 apresenta uma visão que relaciona as várias secções de um documento WSDL, para se entender a estrutura do mesmo e a influência na implementação do serviço. Nesta figura é visível um serviço que aceita pedidos em 3 portas. Duas das portas implementam as operações SOAP do tipo `rpc` e a outra do tipo `document`. Depois são visíveis 3 operações, que definem tipos de portas, compostas por mensagens de entrada e saída. As mensagens são definidas por tipos de dados.

[5] O anexo A.1 apresenta o documento WSDL que define o serviço usado neste projecto de tese.

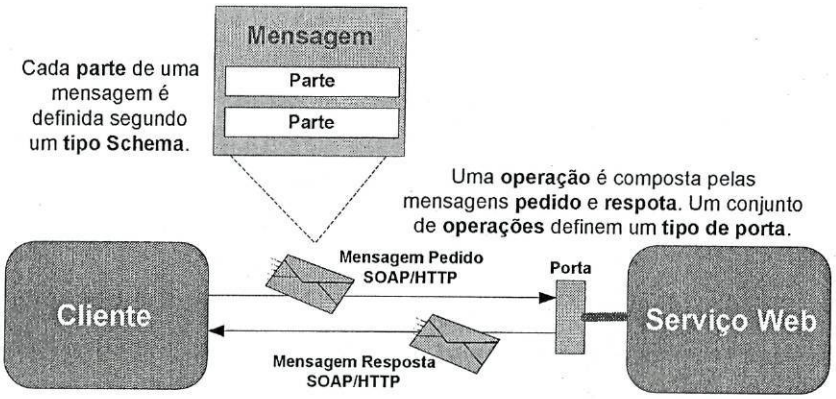



Figura 3.7: Comunicação de um cliente com um serviço Web.

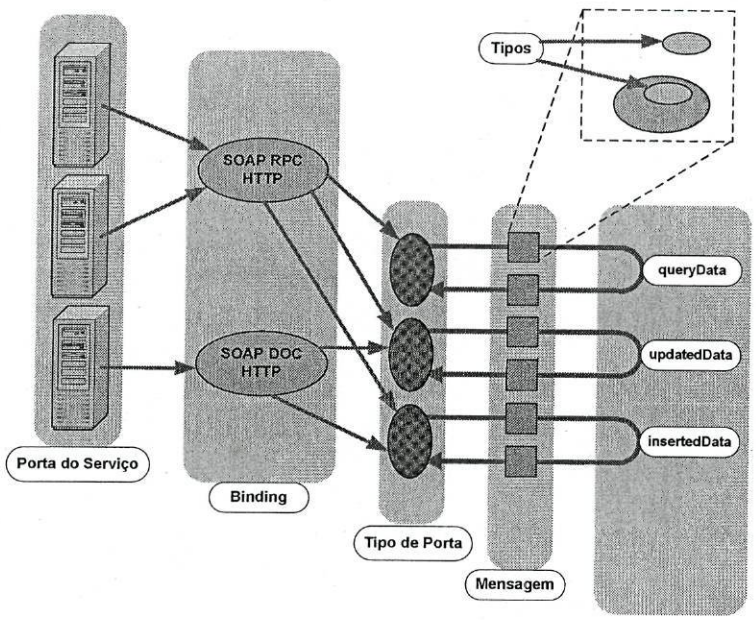



Figura 3.8: Relação das secções de um documento WSDL.

### Definições

Como foi referido, um documento WSDL está dividido em várias secções. Todas elas fazem parte do elemento de topo <definitions>. Este tem um atributo que refere o nome do serviço e outro que indica o *namespace* respectivo. Permite ainda definir outros *namespaces* que são usados na definição do documento. O excerto 3.10 apresenta a secção das definições do documento WSDL.

```
2  <definitions
3      name="DataExchange"
4      targetNamespace="http://www.fc.up.pt/DataExchange"
5      xmlns="http://schemas.xmlsoap.org/wsdl/"
6      xmlns:data="http://www.fc.up.pt/Data-types"
7      xmlns:meta="http://www.fc.up.pt/Meta"
8      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
9      xmlns:tns="http://www.fc.up.pt/DataExchange"
10     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
11      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Programa 3.10: Definições do WSDL.

### Tipos

Normalmente, existe um elemento com o nome <types> que permite definir novos tipos de dados em *Schema*. Estes podem ser usados no documento. Neste documento, o elemento de definição de tipos foi substituído por dois elementos <import> que permitem importar documentos, como é visível no excerto 3.11. Os documentos importados são: a definição de dados obtida na fase anterior do processo e um documento que define um conjunto de elementos que representam meta-informação[6].

```
12      <xsd:import namespace="http://www.fc.up.pt/Data-types"
13                  schemaLocation="http://depa.fc.up.pt:8080/tese/data_definition.xsd"
                        />
14      <xsl:import location="http://depa.fc.up.pt:8080/tese/meta_data.xsd"
15              namespace="http://www.fc.up.pt/Meta"/>
```

Programa 3.11: Importações de definições do WSDL.

### Mensagens

Nesta secção define-se a estrutura das mensagens que são usadas nas operações do serviço. Uma mensagem é constituída por partes. Cada parte tem um nome e um tipo de dados. Os tipos de dados podem ser tipos simples dos *Schemas*, ou então, tipos de dados definidos na secção <types>. No documento foram definidas duas mensagens: a `Request` que define um pedido de informação; a `Reply` define a resposta do pedido. Estas mensagens definem o corpo de uma mensagem SOAP. A mensagem `Request` é vazia, porque considerou-se que o corpo de uma mensagem SOAP apenas pode ter instâncias de objectos dos modelos de dados na resposta referente ao pedido. O excerto 3.12 apresenta a definição das mensagens.

```
16      <message name="Request"/>
17
18      <message name="Reply">
19          <part name="Reply" type="data:SetDataObject"/>
20      </message>
```

Programa 3.12: Definição das mensagens do WSDL.

### Tipos de Porta

Cada tipo de porta define uma operação (função) do serviço. A definição de cada operação envolve a identificação das mensagens que a compõem. A implementação das operações é caracterizada no elemento <binding>. As operações podem ser do tipo:

- **Um-sentido**. Neste tipo de operação é só definida a mensagem que é enviada para o serviço. Não existe mensagem na resposta da comunicação.

[6] A definição de elementos para representar meta-informação vai ser usada no cabeçalho das mensagens SOAP. A sua estrutura vai ser explicada mais à frente.

- **Pedido-resposta**. Neste tipo de operação define-se a mensagem que é enviada para o serviço e a que este envia na resposta ao pedido.

- **Solicitação-resposta**. Neste tipo de operação só é definida a mensagem que o serviço envia na resposta, já que o pedido não envia nenhuma mensagem. É também definida a mensagem que depois é enviada pelo cliente depois de recebida a mensagem do serviço. Esta noção é abstracta, pois tem que ser consultado um *binding* particular para determinar como é que as mensagens são enviadas: como uma única comunicação (com um pedido/resposta HTTP), ou como duas comunicações independentes (como dois pedidos HTTP).

- **Notificação**. Neste tipo de comunicação só é definida a mensagem que o serviço envia na resposta da comunicação, já que o pedido não é composto por nenhuma mensagem.

O WSDL base apenas permite usar os primeiros dois tipos de operações. Para se usar os outros dois é necessários o uso de extensões no elemento <*binding*> associado.

Todas as operações do tipo de porta, que foram definidas neste projecto, são do tipo `Pedido-Resposta`, como apresenta o excerto 3.13. Isto quer dizer que, para que um organismo possa obter objectos de outro, tem de fazer obrigatoriamente um pedido. Todas as operações têm a mensagem `Request` de *input* e a mensagem `Reply` de *output*.

```
21  <portType name="DataExchange">
22      <operation name="queryData">
23      <input message="tns:Request"/>
24          <output message="tns:Reply"/>
25      </operation>
26      <operation name="insertedData">
27      <input message="tns:Request"/>
28          <output message="tns:Reply"/>
29      </operation>
30      <operation name="updatedData">
31      <input message="tns:Request"/>
32          <output message="tns:Reply"/>
33      </operation>
34      <operation name="deletedData">
35      <input message="tns:Request"/>
36          <output message="tns:Reply"/>
37      </operation>
38  </portType>
```

Programa 3.13: Definição do tipo de porta do WSDL.

### Associação

Esta secção define a forma como as operações e as mensagens do serviço são implementadas pelo protocolo de comunicação. Como neste projecto é usado o SOAP, é necessário especificar a gramática do WSDL para o SOAP *binding*, como é visível na excerto 3.14. Este apresenta apenas a definição da operação `queryData`, já que as operações `insertedData`, `updatedData` e `deletedData` têm a mesma definição.

O elemento <`soap:binding`> especifica o tipo de comunicação e o protocolo de transporte. Neste documento é usado o tipo `rpc` e o protocolo de transporte HTTP.

Cada elemento <soap:operation> descreve como é que uma operação é codificada pelo SOAP. Dentro deste, existe o elemento <soap:body>, que descreve a forma como a mensagem da operação é codificada no corpo da mensagem SOAP.

```
40    <wsdl:binding name="DataExchangeBinding" type="tns:DataExchange">
41        <soap:binding style="rpc"
42                      transport="http://schemas.xmlsoap.org/soap/http"/>
43        <wsdl:operation name="queryData">
44          <soap:operation soapAction="queryData"/>
45          <wsdl:input>
46             <soap:body use="encoded"
47                        namespace="urn:xmltoday-delayed-quotes"
48                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
49             <soap:header element="meta:Request"/>
50          </wsdl:input>
51          <wsdl:output>
52             <soap:body use="encoded"
53                        namespace="urn:xmltoday-delayed-quotes"
54                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
55             <soap:header element="meta:Reply"/>
56          </wsdl:output>
57        </wsdl:operation>
58
59        ...
60
61    </wsdl:binding>
```

Programa 3.14: Definição do *binding* do WSDL.

### Serviços

Esta secção permite definir os serviços que o documento WSDL descreve. Cada serviço agrupa um conjunto de portas. Uma porta é um simples ponto definido pela combinação de uma associação e de um endereço. O endereço depende do protocolo de comunicação definido na respectiva associação. Neste caso é usado o SOAP como demonstra o excerto 3.15.

```
105   <service name="DataExchange_Service">
106       <port binding="tns:DataExchangeBinding" name="urn:DataExchange">
107           <soap:address location="http://depa.fc.up.pt:8080/axis/services/
                  DataExchange"/>
108       </port>
109   </service>
```

Programa 3.15: Definição do serviço em WSDL.

## 3.3.2 A meta-informação

Durante a apresentação da definição do serviço segundo o documento WSDL, foram referidas duas importações de definições de dados: a definição de dados obtida a partir do modelo de dados e a definição de meta-informação. A importação da primeira definição tem como objectivo definir o corpo das mensagens SOAP. Portanto o corpo dessas mensagens apenas pode conter instâncias dos objectos, definidos no modelo de dados de acordo com o *Schema*.

Como as mensagens SOAP são documentos XML e foi assumido que o corpo das mensagens só pode ter instâncias de objectos, é necessária uma forma de caracterização das mensagens. Como é que se sabe se a mensagem é um pedido ou uma resposta? No caso de ser um pedido, o que é que ela

pretende obter como resposta? Tem de existir informação que caracterize a mensagem. Por exemplo, se uma mensagem for relativa a um pedido tem de ter um identificador, um *timestamp*, que objectos é que se pretende obter, etc. Tudo isto é considerado meta-informação.

Tirando partido da estrutura das mensagens SOAP, foi definido um formato de dados que permite especificar a informação necessária para caracterizar as mensagens. Esse formato de dados corresponde à segunda definição que é importada pelo documento WSDL. O conjunto de meta-informação enviada no cabeçalho do documento, para caracterizar as mensagens, é descrito na tabela 3.3. Essa informação depende do tipo de documento se é um pedido ou uma resposta.

| Elemento | Pedido | Resposta | Descrição |
|---|---|---|---|
| **Timestamp** | sim | sim | Informação relativa à data e hora da mensagem. É informação obrigatória. |
| **Source** | sim | sim | Informação relativa à origem da mensagem. Não existe uma restrição em relação a esta informação. Serve apenas para indicar a fonte de informação. É informação obrigatória. |
| **ID** | sim | sim | A mensagem tem de ter um identificador. Isto permite identificar de forma única cada mensagem enviada de qualquer local. Mais uma vez não existe uma restrição em relação ao identificador propriamente dito. É informação obrigatória. |
| **SinceDate** | sim | não | Isto é informação opcional. Permite restringir a obtenção de informação relativamente a uma data. |
| **RequestID** | não | sim | As mensagens de resposta têm que indicar o pedido correspondente. |

Tabela 3.3: Informação do cabeçalho das mensagens SOAP.

As mensagens pedidos permitem identificar as classes de objectos que se pretende obter informação através dos elementos apresentados na tabela 3.4. O elemento <Object> pode ser utilizado de forma isolada, enquanto que o <Query> tem que ser usado em conjunto com o <Object>.

O documento 3.16 apresenta a definição correspondente à meta-informação enviada no cabeçalho das mensagens SOAP dos pedidos e das respostas.

```
1  <?xml version="1.0"?> <xsd:schema
2      xmlns="http://www.w3.org/2001/XMLSchema"
3          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4          targetNamespace="http://www.fc.up.pt/Meta"
5          xmlns:meta="http://www.fc.up.pt/Meta"
6          elementFormDefault="qualified" attributeFormDefault="unqualified">
7
8        <xsd:complexType name="Request">
9          <xsd:sequence>
10           <xsd:element name="Object" type="xsd:string" minOccurs="1"
                 maxOccurs="1"/>
11           <xsd:element name="Query" type="xsd:string" maxOccurs="1"/>
12         </xsd:sequence>
13         <xsd:attribute name="timestamp" type="xsd:string" use="required"/>
```

```
14              <xsd:attribute name="source" type="xsd:string" use="required"/>
15              <xsd:attribute name="ID" type="xsd:string" use="required"/>
16              <xsd:attribute name="sinceDate" type="xsd:string" use="optional"/>
17          </xsd:complexType>
18
19          <xsd:complexType name="Reply">
20              <xsd:attribute name="timestamp" type="xsd:string" use="required"/>
21              <xsd:attribute name="source" type="xsd:string" use="required"/>
22              <xsd:attribute name="ID" type="xsd:string" use="required"/>
23              <xsd:attribute name="RequestID" type="xsd:string" use="required"/>
24          </xsd:complexType>
25  </xsd:schema>
```

Programa 3.16: *Schema* que define meta-informação das mensagens usado pelo documento WSDL.

| Elemento | Descrição |
|---|---|
| **Object** | Identifica a classe de objectos que se pretende obter. |
| **Query** | Expressão XPath que tem o papel de filtro a aplicar sobre o conjunto de objectos da classe pretendida, para obter apenas um sub-conjunto desses objectos. Esta expressão é aplicada sobre um documento com todos os objectos da classe pretendida. |

Tabela 3.4: Elementos que identificam os objectos pretendidos.

### 3.3.3 Exemplo

#### Implementação do serviço

Seguidamente é descrita a implementação do serviço Web segundo o documento WSDL apresentado, destacando os critérios usados na escolha das tecnologias utilizadas. A implementação das aplicações servidor e cliente necessitam do documento WSDL, como é visível na figura 3.9. Depois, é descrita a implementação do serviço, e por fim é apresentada a aplicação cliente.

#### Tecnologias utilizadas

A tecnologia escolhida para implementar o serviço foi o Java. As razões que levaram à sua escolha são as seguintes:

- **Independência**. É uma linguagem independente do sistema e da plataforma.
- **Tipos**. É uma linguagem fortemente tipada.
- **Utilitários**. É uma linguagem muito utilizada. Existem bastantes utilitários Java que podem ser usados na implementação do serviço.

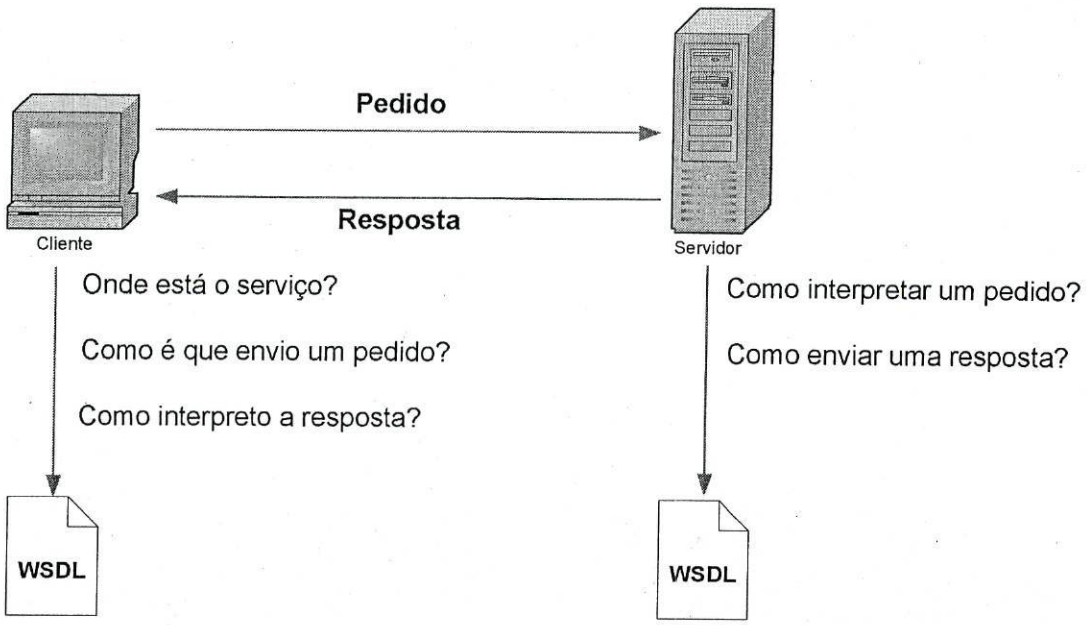



Figura 3.9: Necessidade do documento WSDL por parte do cliente e do servidor.

Para facilitar a implementação do serviço foi usado o *Axis 1.0* do *Apache Project*. Este projecto é uma continuação do projecto *Apache SOAP*. O *Axis* é um conjunto de classes Java que facilitam a construção e utilização de documentos SOAP. Esse pacote de classes disponibiliza também algumas funcionalidades muito úteis à construção de serviços. Por exemplo, a classe WSDL2Java permite criar o esqueleto das classes Java da aplicação do serviço a partir do documento WSDL[AXI, TOM].

O resultado da geração de classes através do WSDL2Java é um *servlet*. Este tipo de classe Java necessita de condições especificas para poder ser executado. Os *servlets* são a tecnologia Java para extender as funcionalidades dos servidores Web. Proporcionam métodos independentes da plataforma e do servidor para criar aplicações baseadas na Web, sem as limitações apresentadas pelos CGI's. Esta independência permite que a escolha do servidor, plataforma e ferramentas fique em aberto para decisão posterior[SER].

Como os *servlets* são executados por servidores Web, o *Tomcat 4* do *Jakarta Project* foi usado para cumprir essa função. Assim, a um serviço fica associado um *servlet*. Quando a aplicação cliente envia uma mensagem HTTP para o servidor Web, este sabe qual o serviço a executar, porque essa informação faz parte do cabeçalho da mensagem HTTP.

O resultado da execução do *servlet* relativo ao serviço é uma mensagem SOAP. Esta mensagem é passada para o servidor HTTP. Por sua vez, o servidor adiciona o cabeçalho HTTP à mensagem SOAP e envia para o cliente que fez o pedido.

#### 3.3.3.1 Servidor

O processo de implementação do serviço começa por utilizar o WSDL2Java, uma das ferramentas disponibilizadas pelo *Axis*. Esta ferramenta permite criar um esqueleto das classes do servidor e do cliente. Das classes que foram geradas para o servidor, só a classe `DataExchangeBindingImpl.java` requer alterações por parte do programador para implementar as operações do serviço. Esta classe

tem como prefixo `DataExchange` que corresponde ao nome do serviço definido no documento WSDL.

Para colocar o serviço a funcionar é necessário fornecer informações do serviço ao *Tomcat* como sejam: o nome do serviço, as operações do serviço, o nome do *servlet* a executar. Um excerto dessa informação é apresentado em 3.17. O *Axis* dispõem de uma ferramenta que coloca o conteúdo do ficheiro *deploy.wsdd*[7], no ficheiro que contem a informação que o *Tomcat* necessita para os serviços.

```
14      xmlns="http://xml.apache.org/axis/wsdd/"
15      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
16
17    <!-- Services from DataExchange_Service WSDL service -->
18
19   <handler name="track" type="java:samples.tese.LogHandler">
20    <parameter name="filename" value="MyService.log"/>
21   </handler>
22
23    <service name="urn:DataExchange" provider="java:RPC">
24    <requestFlow>
25     <handler type="track"/>
26    </requestFlow>
27        <parameter name="className" value="samples.tese.DataExchangeBindingImpl"/>
28        <operation name="queryData" returnQName="Reply" >
29        </operation>
30        <operation name="insertedData" returnQName="Reply" >
31        </operation>
32        <operation name="updatedData" returnQName="Reply" >
33        </operation>
34        <operation name="deletedData" returnQName="Reply" >
35        </operation>
36        <parameter name="allowedMethods" value="queryData insertedData
              updatedData deletedData"/>
37        <parameter name="scope" value="Session"/>
38
39       ...
40
41  </service>
```

Programa 3.17: Informação sobre o serviço disponibilizada ao servidor HTTP

O anexo A.2.1 apresenta o servidor do serviço, explicando cada um dos procedimentos que o compõem:

- **queryData**, **insertedData**, **updatedData**, **deletedData**. Procedimentos correspondentes às operações.
- **getHeaderInfo**. Trata a meta-informação do cabeçalho da mensagem SOAP pedido.
- **buildReply**. Cria o documento que vai constituir o corpo da mensagem SOAP de resposta.
- **doQuery**. Filtra um conjunto de elementos do conjunto de todos os objectos de uma classe.
- **createRootElement**. Coloca os elementos dos objectos que irão ser passados no documento de retorno, dentro do elemento `setDataObject`.
- **setHeaderInfo**. Procedimento que coloca a meta-informação no cabeçalho da mensagem SOAP de resposta.

[7]Este ficheiro é uma especificação WSDD (*Web Service Deployment Descriptor*). É uma especificação XML usada para disponibilizar serviços Web

### Cliente

A aplicação cliente destina-se a pedir informação aos serviços, em que os pedidos são feitos através de mensagens SOAP enviadas ao servidor. A aplicação cliente depende bastante dos objectivos definidos pelo organismo. Pode ser uma aplicação de execução na linha de comandos, pode ser uma aplicação em ambiente gráfico, pode fazer parte de outras aplicações, etc.

Como foi referido, a aplicação cliente pode ser criada a partir das classes geradas pela ferramenta WSDL2Java do *Axis*.

A aplicação que foi criada no âmbito deste projecto, obriga os pedidos de informação a especificar: o tipo de pedido e a classe dos objectos de que se pretende obter informação. Os tipos de pedido suportados são os seguintes:

- **queryData**. Pedido de informação de instâncias de uma classe do sistema.
- **insertedData**. Pedido de informação de instâncias de uma classe. Este tipo de pedido pode ser usado em conjunto com a restrição de uma data para obter os objectos que foram inseridos a partir de uma determinada data.
- **updatedData**. Pedido de informação de instâncias de uma classe que foram alteradas. É usado como pedido anterior.
- **deletedData**. Pedido de informação de instâncias de uma classe que foram eliminadas. É usado como nos dois pedidos anteriores.

Os parâmetros extra que podem ser adicionados ao pedido de informação, são os seguintes:

- **XPath**. É uma expressão XPath que deve ser aplicada sobre o documento com os objectos da classe especificada. Em primeiro lugar o serviço cria um documento com os objectos da classe e depois aplica esta expressão como filtro.
- **Data**. Pode ser especificada uma data de restrição dos objectos.

Existem ainda três parâmetros extra que podem ser usados na execução da aplicação cliente, mas que não fazem parte da meta-informação que é passada na mensagem SOAP pedido. Isto é, são funcionalidades do cliente que só são consideradas quando a mensagem de resposta chega ao cliente.

- **XML**. Permite guardar a resposta como um ficheiro XML.
- **HTML**. Permite guardar a resposta como um ficheiro HTML.
- **SQL**. Permite guardar a resposta como um ficheiro com instruções SQL. Esta opção não funciona com o tipo de pedido `queryData`.

Para guardar a informação nos formatos HTML ou SQL são usadas folhas de estilo XSLT que são aplicadas sobre o documento XML que é recebido como resposta ao pedido. A execução da aplicação sem quaisquer parâmetros apresenta ao utilizador a forma de utilização da mesma, como é visível na figura 3.18.

O anexo A.2.2 apresenta a aplicação cliente dando uma explicação dos seus procedimentos:

```
bash-2.05# java samples.tese.Client No arguments Usage:  java
sample.tese.Client options [extra] Options:
  -m method          Method Name: [queryData|insertedData|updatedData|deletedData]
  -O object          Data Object

Extra:
  -q xpath           Document filter with a XPATH expression (only works with "
     queryData" method)
  -d date            Data since date. Date Format: yyyyMMddhhmmss
  -lx                Log result as XML
  -lh                Log result as HTML
  -ls                Log result as SQL (only works with "insertedData","updatedData
     ","deletedData" methods)
  -?                 Display this message
```

Programa 3.18: Utilização da aplicação cliente.

- **doMain**. Trata dos parâmetros passados de execução da aplicação
- **setServiceDefault**. Define o serviço a invocar e o local do mesmo.
- **doCall**. Invoca o serviço. Isto é, envia uma mensagem SOAP para o serviço e recebe uma mensagem como resposta.
- **parseReply**. Verifica sintacticamente o documento do corpo da mensagem SOAP resposta, segundo o *Schema* de definição de dados, através de um *parser* XML.
- **logXMLFile**. Grava o documento resultado como um ficheiro XML.
- **logHTMLFile**. Usa uma folha de estilos para aplicar sobre o documento resultado e grava a informação no formato HTML num ficheiro.
- **logSQLFile**. Usa uma folha de estilos para aplicar sobre o documento resultado e grava a informação no formato SQL num ficheiro. Esta opção apenas funciona com as operações *insertedData*, *updatedData*, *deletedData*.

O comando 3.19 exemplifica a utilização da aplicação cliente desenvolvida para obter informações dos Alunos, em que 'João' faz parte do nome. O resultado deve ser guardado em XML e HTML e apresentado em 3.20 e 3.21 respectivamente.

```
java samples.tese.Client -O Aluno -m queryData -q
"/Aluno[contains(nome,'João')]"
```

Programa 3.19: Utilização da aplicação para fazer um pedido de informação.

```
1  <?xml version="1.0" encoding="UTF-8"?> <data:SetDataObject
2      xmlns:data="http://www.fc.up.pt/Data-types"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.fc.up.pt/Data-types
5      http://depa.fc.up.pt:8080/tese/data_definition.xsd">
6      <data:Aluno>
7       <data:ncd>950311008</data:ncd>
8       <data:nome>João Pedro Fonseca d</data:nome>
9      </data:Aluno>
10     <data:Aluno>
11      <data:ncd>930307027</data:ncd>
12      <data:nome>João Filipe Carvalho</data:nome>
13     </data:Aluno>
14     <data:Aluno>
```

```
15        <data:ncd>880307019</data:ncd>
16        <data:nome>João Manuel Gonçalve</data:nome>
17       </data:Aluno>
18       <data:Aluno>
19        <data:ncd>920301033</data:ncd>
20        <data:nome>Maria João Coelho Pi</data:nome>
21       </data:Aluno>
22       <data:Aluno>
23        <data:ncd>920307019</data:ncd>
24        <data:nome>João Miguel Martins</data:nome>
25       </data:Aluno>
26       <data:Aluno>
27        <data:ncd>960311005</data:ncd>
28        <data:nome>David João de Castro</data:nome>
29       </data:Aluno>
30       <data:Aluno>
31        <data:ncd>960311035</data:ncd>
32        <data:nome>Maria João Fonseca A</data:nome>
33       </data:Aluno>
34       <data:Aluno>
35        <data:ncd>940307048</data:ncd>
36        <data:nome>Nelson João Rodrigue</data:nome>
37       </data:Aluno>
38       <data:Aluno>
39        <data:ncd>950311009</data:ncd>
40        <data:nome>João Pedro Leite Cor</data:nome>
41       </data:Aluno>
42       <data:Aluno>
43        <data:ncd>910307010</data:ncd>
44        <data:nome>João Carlos de Sousa</data:nome>
45       </data:Aluno>
46       <data:Aluno>
47        <data:ncd>940311017</data:ncd>
48        <data:nome>João Sá dos Reis Fre</data:nome>
49       </data:Aluno>
50       <data:Aluno>
51        <data:ncd>960307023</data:ncd>
52        <data:nome>João Ricardo Ferreir</data:nome>
53       </data:Aluno>
54       <data:Aluno>
55        <data:ncd>000311005</data:ncd>
56        <data:nome>João Henrique Louro</data:nome>
57       </data:Aluno>
58       <data:Aluno>
59        <data:ncd>010365007</data:ncd>
60        <data:nome>João Brandão Reis</data:nome>
61       </data:Aluno>
62       <data:Aluno>
63        <data:ncd>010365008</data:ncd>
64        <data:nome>João Emanuel Rodrigu</data:nome>
65       </data:Aluno>
66       <data:Aluno>
67        <data:ncd>830301032</data:ncd>
68        <data:nome>Maria João da Silva</data:nome>
69       </data:Aluno>
70       <data:Aluno>
71        <data:ncd>000307064</data:ncd>
72        <data:nome>Vera João Novais Gom</data:nome>
73       </data:Aluno>
74       <data:Aluno>
75        <data:ncd>990365010</data:ncd>
76        <data:nome>Mario João Gonçalves</data:nome>
77       </data:Aluno>
78      </data:SetDataObject>
```

Programa 3.20: Documento de resposta ao pedido.

```
1  <html xmlns:data="http://www.dataexchangeservice.com/Data-types">
2  <head>
3  <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
4  <title></title>
5  </head>
6  <body>
7  <h1>Result Data</h1>
8  <h3>Aluno</h3>
9  <table border="1">
10 <tr>
11 <th align="left">ncd</th><th align="left">nome</th>
12 </tr>
13 <td>950311008</td><td>João Pedro Fonseca d</td>
14 <tr>
15 <td>930307027</td><td>João Filipe Carvalho</td>
16 </tr>
17 <tr>
18 <td>880307019</td><td>João Manuel Gonçalve</td>
19 </tr>
20 <tr>
21 <td>920301033</td><td>Maria João Coelho Pi</td>
22 </tr>
23 <tr>
24 <td>920307019</td><td>João Miguel Martins</td>
25 </tr>
26 <tr>
27 <td>960311005</td><td>David João de Castro</td>
28 </tr>
29 <tr>
30 <td>960311035</td><td>Maria João Fonseca A</td>
31 </tr>
32 <tr>
33 <td>940307048</td><td>Nelson João Rodrigue</td>
34 </tr>
35 <tr>
36 <td>950311009</td><td>João Pedro Leite Cor</td>
37 </tr>
38 <tr>
39 <td>910307010</td><td>João Carlos de Sousa</td>
40 </tr>
41 <tr>
42 <td>940311017</td><td>João Sá dos Reis Fre</td>
43 </tr>
44 <tr>
45 <td>960307023</td><td>João Ricardo Ferreir</td>
46 </tr>
47 <tr>
48 <td>000311005</td><td>João Henrique Louro</td>
49 </tr>
50 <tr>
51 <td>010365007</td><td>João Brand&atilde;o Reis</td>
52 </tr>
53 <tr>
54 <td>010365008</td><td>João Emanuel Rodrigu</td>
55 </tr>
56 <tr>
57 <td>830301032</td><td>Maria João da Silva</td>
58 </tr>
59 <tr>
60 <td>000307064</td><td>Vera João Novais Gom</td>
61 </tr>
62 <tr>
63 <td>990365010</td><td>Mario João Gonçalves</td>
64 </tr>
65 </table>
66 </body>
```

```
67 </html>
```

Programa 3.21: Resposta ao pedido em HTML.

### Adaptar a aplicação a novos serviços

O serviço implementado pode ser usado para qualquer definição de dados. Para isso é necessário usar o documento WSDL definido, fazendo alteração das referências a documentos, alteração do URL do serviço e alteração do nome do serviço. Depois é executado o WSDL2Java para criar o esqueleto do serviço. Das classes Java geradas, a que termina com `BindingImpl.java` é a única que necessita de ser implementada. Desta forma, esta classe pode ser substituída pela implementada `DataExchangeBindingImpl.java`, sendo ainda necessário alterar o nome da classe para o nome do serviço seguido de *BindingImpl*[8].

Em relação à aplicação cliente, é apenas necessário alterar o URL e o nome do serviço no ficheiro `Client.java`. Isso é feito alterando as linhas de código apresentadas no excerto 3.22, no método `setServiceDefault`.

```
1        setServiceName("urn:DataExchange");
2        setServiceURL("http://depa.fc.up.pt:7080/axis/servlet/AxisServlet");
```

Programa 3.22: Instruções que definem o URL e nome do serviço.

### A segurança no protótipo

Com foi referido, foi usado o *Axis* para facilitar escrita e leitura das mensagens SOAP. Dele faz parte um conjunto de classes que permitem usar assinaturas digitais e cifras sobre as mensagens SOAP. O *Axis* segue a especificação WS-Security (*Web Services Security*), desenvolvida pela IBM, Microsoft e pela VeriSign. O objectivo desta implementação é extender a especificação do cabeçalho das mensagens SOAP para adicionar elementos usados pelas metodologias de segurança. Em seguida é apresentado WS-Security e o XKMS. Por último é descrita a forma de adicionar segurança usando o *Axis*.

### WS-Security

A falta de segurança dos serviços Web compromete significativamente a sua utilização. Para contornar esses problemas de segurança foi então criado o WS-Security[Aps02]. Este protocolo suporta, integra e unifica vários modelos, mecanismos e tecnologias de segurança anteriormente descritas.

O WS-Security integra um conjunto *standard* de extensões SOAP que podem ser usadas para adicionar metodologias de segurança aos serviços Web, garantindo integridade e confidencialidade das mensagens. Permite ainda autenticar mensagens através de assinaturas digitais, colocadas no cabeçalho das mensagens SOAP.

[8] O uso da aplicação implementada requer ainda algumas alterações dos nomes de classes, no caso do nome do novo serviço ser diferente.

O WS-Security não descreve explicitamente nenhum protocolo de segurança. Como todos os protocolos de segurança garante que os protocolos construídos sobre o WS-Security não são vulneráveis a ataques. Em seguida é apresentada a forma como o WS-Security trata os problemas de segurança.

Quando se adiciona segurança aos serviços Web devem estar sempre presentes as seguintes considerações:

- A mensagem SOAP não pode ser vista ou alterada por terceiros.
- O serviço deve implementar medidas de segurança no processamento da mensagem que impeça o envio de mensagens por pessoas não autorizadas.

O modelo de segurança das mensagens foi definido para tratar destes problemas. O WS-Security especifica um modelo abstracto de segurança das mensagens através de *tokens* de segurança com assinaturas digitais. A sua especificação não está restrita a um tipo particular de *token*. Neste contexto, um *token* de segurança é uma representação de informação relativa à segurança. Esta pode-se manifestar de diversas formas como certificados X.509, bilhetes/autenticadores Kerberos, uma credencial utilizador/password, etc. Qualquer que seja a opção usada, os programadores estão isolados dos detalhes de segurança do WS-Security. O WS-Security permite a adição de vários *tokens* para a mesma mensagem aumentando os níveis de segurança.

As principais considerações de segurança do WS-Security são a confidencialidade e a integridade. O WS-Security proporciona formas de protecção através de cifras e/ou assinaturas digitais do corpo, cabeçalho e anexos das mensagens.

A integridade das mensagens é garantida através do XML-SIG em conjunto com os *tokens* de segurança. Os mecanismos de integridade foram criados de forma a poderem suportar múltiplas assinaturas, possivelmente aplicadas por vários actores, e para ser extensível, podendo assim suportar novos formatos de assinaturas.

A confidencialidade das mensagens é conseguida usando o XML-ENC em conjunto com os *tokens* de segurança. Um dos objectivos na criação destes mecanismos é deixar em aberto a possibilidade de usar outras formas de cifra.

O receptor da mensagem pode rejeitar ou responder com uma mensagem de erro, quando recebe uma mensagem em que a assinatura digitar é inválida, ou em falta, ou então diga respeito a um pedido desapropriado.

O WS-Security é baseado em *standards* e especificações existentes. Não especifica como é que os mecanismos de segurança, que requerem múltiplas trocas de informação, são estabelecidos, nem refere nada relativamente às chaves. Em seguida é apresentado o XKMS que é responsável pelas chaves usadas nas metodologias de segurança.

### XKMS

A criptografia de chave pública permite estabelecer comunicações seguras entre quaisquer organismos, desde que cada uma obtenha as chaves publicas dos restantes a partir de um local de confiança. A PKI (*Public Key Infrastructure*) proporciona uma forma segura de trocar informação relativa a chaves publicas, baseando-se em certificados digitais. A configuração do PKI é extremamente complexa, já

que tem de reflectir ao mesmo tempo a complexidade e a subtileza dos relacionamentos de confiança entre organismos reais[PM].

Para que o PKI seja útil tem de ser usado por todos os organismos intervenientes, como é visível na figura 3.10. Isto limitou a utilização e evolução do PKI, já que necessitava de uma complexa implementação por parte de cada organismo. As actualizações dos PKI's eram difíceis de fazer, pois para além de serem feitas em cada organismo, dependia muito da configuração usada por cada um.

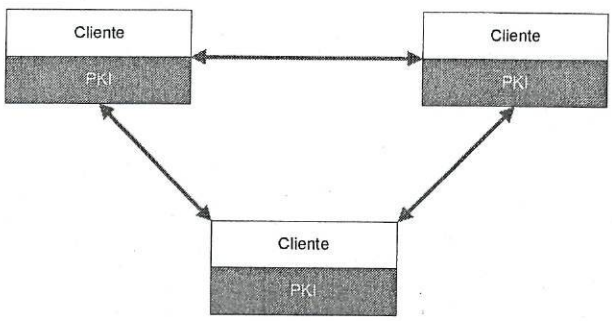



Figura 3.10: Complexidade do uso dos PKI's iniciais.

A introdução do *Trust Services* veio libertar os organismos da complexa gestão dos PKI's, já que essa gestão passou a ser feita nestas entidades específicas. Estes serviços asseguram que todos os organismos da mesma organização suportam as funcionalidades do PKI, já que estas deixam de residir nos organismos passando para os *Trust Service*, como descreve a figura 3.11.

O XKMS (*XML Key Management Specification*) é um PKI *Trust Services*. As suas vantagens são as seguintes:

- Retira trabalho aos organismos.
- A sintaxe XML simplifica a implementação.
- As relações de confiança são configuradas a este nível.
- A adição de novas funcionalidades da PKI é feita num único local.

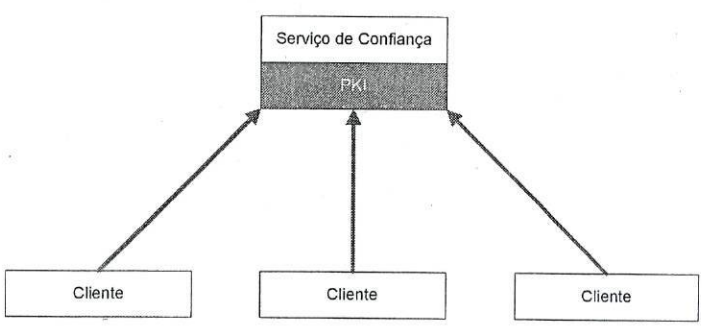



Figura 3.11: Uso dos PKI's com serviços de confiança.

O XKMS consiste em duas partes, o X-KISS (*XML Key Information Service Specification*) e o X-KRSS (*XML Key Registration Service Specification*). O X-KISS define o protocolo de um serviço de

confiança que resolve informação relativa a chaves publicas em elementos <KeyInfo> do XML-SIG. O X-KRSS actua como um serviço de registo de chaves publicas.

#### 3.3.3.2 Implementação de segurança com o Axis

O *Axis* é uma extensão do Apache SOAP. Para além de novas funcionalidades, proporciona uma plataforma que funciona em cadeia. Uma cadeia consiste numa sequência de transformadores, isto é, programas invocados antes ou depois da execução do serviço. Os transformadores permitem processar a mensagem de forma isolada do serviço. As cadeias podem ser usadas do lado do cliente e do lado do serviço.

Os transformadores podem ser usados para um ou mais serviços. Estes podem ser alterados sem a necessidade de alteração do serviço ou da aplicação cliente. Podem ser usados vários tipos de *handlers* no mesmo serviço. Por exemplo, um responsável pela assinatura digital e outro responsável pela cifra.

No projecto foram usados dois transformadores, um do lado do cliente e outro do lado do servidor. Desta forma é usado um único programa para o cliente assinar as mensagens enviadas para o serviço verificar essa assinatura.

Uma das tarefas mais complicadas é a troca de chaves entre o cliente e o serviço. Para isso pode ser usado o XKMS. A obtenção das chaves pode ser feita a partir de um PKI, contudo, foram usadas chaves que fazem parte de um exemplo da ferramenta *Axis*. As chaves residem num ficheiro local ao servidor e ao cliente.

Durante o desenvolvimento da aplicação de exemplo foi implementado um mecanismo de segurança usando o WS-Security do *Axis*. Desse mecanismo apenas foram usadas assinaturas digitais.

Como foi referido, esta forma de suportar mecanismos de segurança, usando transformadores, é isolada das aplicações. Assim, o processo de adição e remoção desses mecanismos torna-se muito flexível.

**O serviço**

Para disponibilizar um serviço com o *Axis* é necessário executar o comando 3.23, tendo como argumento um ficheiro do tipo WSDD. Este contem informação relativa ao serviço, nomeadamente: a classe que corresponde ao serviço, URL do serviço, etc.

```
java org.apache.axis.client.AdminClient
samples/tese/securitydeploy.wsdd
```

Programa 3.23: Comando para disponibilizar o serviço.

Para usar um transformador associado ao serviço é necessário fazer uma referência ao mesmo, no ficheiro WSDD. Essa referência indica qual a classe que tem o papel de transformador, como é visível em 3.24.

```
1 <handler name="track"
2          type="java:samples.tese.LogHandler">
3   <parameter name="filename" value="MyService.log"/>
```

```
4 </handler>
```

Programa 3.24: Documento que refere o *handler* do lado do servidor.

O transformador que foi usado foi o do exemplo da ferramenta *Axis*. A sua única função é verificar a assinatura da mensagem que chega ao serviço, segundo um ficheiro chamado `keystore.jks`.

Depois de verificada a assinatura da mensagem, esta passa para o serviço propriamente dito.

### Cliente

A assinatura das mensagens enviadas pelo cliente a um serviço, é activada usando um transformador, como acontece com o servidor, mas com o comando expresso na figura 3.25.

```
java org.apache.axis.utils.Admin
clientclientsecuritydeploy.wsdd
```

Programa 3.25: Comando para disponibilizar o *handler* do lado do cliente.

No ficheiro `clientsecuritydeploy.wsdd` tem de existir uma referência à classe com o papel de transformador e uma referência ao ficheiro que contem a chave de assinatura, como exemplifica a figura 3.26. É usada a classe `ClientSigningHandler` e o ficheiro `keystore.jks`, respectivamente. O transformador usado foi o do exemplo da ferramenta *Axis*.

```
1  <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2              xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3   <globalConfiguration>
4    <requestFlow>
5     <handler type="java:samples.tese.ClientSigningHandler">
6      <parameter name="keystore" value="samples/tese/keystore.jks"/>
7     </handler>
8    </requestFlow>
9   </globalConfiguration>
10 </deployment>
```

Programa 3.26: Documento que faz a referência à chave.

Quando a aplicação cliente envia a mensagem, esta é passada para o transformador. Por sua vez, este vai buscar a chave ao ficheiro e assina a mensagem. Depois é que a mensagem é enviada ao serviço.

# Capítulo 4

# Conclusão

Como conclusão a este trabalho é feita em primeiro lugar uma análise crítica dos objectivos propostos. Depois é feita uma análise da implementação enumerando os principais problemas e vantagens da utilização da metodologia. Em seguida são apresentadas algumas ideias futuras para o trabalho e que inovações das tecnologias utilizadas trariam vantagem à metodologia. Para finalizar é dada uma opinião pessoal do trabalho.

## 4.1 Análise da metodologia

Neste trabalho foi proposta uma metodologia para cumprir os seguintes objectivos:

- Obter uma especificação do sistema de informação global de uma organização.
- Definir relações de partilha de dados entre os organismos.
- Definir direitos e responsabilidades dos organismos.
- Criar uma definição de dados independente do sistemas de informação dos organismos.
- Definir um modelo de comunicação para a partilha de dados.

O processo de modelação proposto consegue definir a base do sistema de informação global da organização identificando o papel dos organismos. Os modelos de dados segundo diagramas de classes UML conseguem representar os dados, conseguem identificar a proveniência dos mesmos e também os direitos e responsabilidades de cada organismo a este nível. Neste processo é aconselhada a existência de uma entidade reguladora que garanta a aplicação das definições de dados produzidas pela metodologia por parte dos organismos envolvidos na partilha de dados.

Do modelo de dados podemos obter uma definição de dados (*Schema*) por cada organismo. Esta permite diminuir o número de conversores necessários N*N-N com N organismos, para 2N. A obtenção de uma definição por organismo também tem a vantagem de cada organismo ter associado apenas definições do seu âmbito e não do sistema global.

Na etapa da comunicação de dados entre os organismos, a metodologia baseia-se mais uma vez no uso de *standards* deixando em aberto a escolha da plataforma, sistema e linguagem de implementação

dos serviços e dos clientes. Nesta etapa foram também analisadas algumas questões de segurança. No entanto, este é um tema ainda ainda em estudo e não existem garantias absolutas da eficiência das soluções apresentadas. Este é um dos motivos que condiciona a utilização de serviços Web para a comunicação de dados.

Um dos objectivos, a definição dos direitos e responsabilidades dos organismos, não foi completamente conseguido. Essa informação apenas é representada ao nível do modelo de dados, não se reflectindo nas etapas posteriores. Contudo, foi referido que deve existir uma entidade reguladora para obrigar os organismos a cumprirem esses direitos e responsabilidades ao nível da implementação dos serviços.

## 4.2 Análise da implementação

### Utilidade da metodologia

A concretização da utilização desta metodologia permitiu tirar algumas conclusões bastante positivas e outras negativas. No entanto a conclusão geral a que chegamos é que resolve o problema da comunicação de dados entre organismos heterogéneos.

Uma organização composta por diversos organismos com sistemas de informação heterogéneos e que necessitem de partilhar informação podem usar esta metodologia para resolver esse processo que é sempre bastante complexo. Para isso podem seguir as sugestões de ferramentas que são apresentadas ou então tirar partido de ferramentas e plataformas que permitam um maior desempenho e facilidade de implementação na solução.

### Dificuldades encontradas.

Na implementação desta metodologia fomos confrontados com algumas adversidades. Para podermos definir o processo de modelação de acordo com o primeiro objectivo a que nos propusemos foram criados vários modelos de dados em diferentes ferramentas de modelação UML. Durante esta tarefa foram testadas algumas ferramentas gratuitas e verificamos que não consideravam toda a especificação UML (pelo menos as funcionalidades).

A solução foi utilizar as ferramentas de demonstração disponibilizadas pelas principais empresas de desenvolvimento desta área como o RationalRose e o MagicDraw que consideram toda a especificação UML. No entanto, existe um problema em usar ferramentas de demonstração comerciais. Estas versões estão limitadas a um determinado número de componentes, restringindo o modelo que se pretendia criar 3 organismos com 25 classes.

Outra dificuldade encontrada nas ferramentas UML durante o processo de modelação foi o seu baixo nível de validação. Embora uma das principais características do UML seja a extensibilidade, algumas ferramentas permitem criar modelos violando regras da especificação do UML. Isto obriga o utilizador a ter um conhecimento profundo da especificação UML para criar modelos correctos, isto é, sem ultrapassar os limites da especificação.

O modelo de comunicação escolhido (serviços Web) permitiu escolher uma ferramenta para implementar as aplicações cliente e servidor. Existem diversas ferramentas que facilitam a criação da

definição WSDL e do esqueleto das aplicações cliente e servidor, e também disponibilizam outras funcionalidades e documentação, mas são geralmente ferramentas comerciais. A opção por ferramentas gratuitas obrigou a escrever o documento WSDL na sua totalidade. Foi usada a ferramenta *Axis*, para gerar o esqueleto das classes Java das aplicações (usando a funcionalidade WSDL2Java) e depois para facilitar a construção de documentos SOAP.

Outra dificuldade encontrada foi a implementação das aplicações usando a ferramenta *Axis*. Este tipo de ferramentas estão ainda em fase de desenvolvimento e a sua documentação é extremamente limitada. Foram testados vários exemplos disponibilizados pelo *Axis* e alguns não funcionavam. Tivemos de recorrer muitas vezes às *mailing lists* da ferramenta para tentar resolver alguns problemas recorrendo à experiência de outros utilizadores.

### Vantagens da utilização da metodologia

A implementação permitiu verificar as seguintes vantagens na utilização desta metodologia:

- A facilidade de modelação em ferramentas comerciais. Estas permitem enumeras funcionalidades que facilitam o processo de modelação. São capazes de criar modelos numa escala bastante superior à que foi usada no exemplo.
- A eficácia do processo de mapeamento do modelo de dados nas definições de dados. No exemplo, este processo foi muito rápido (menos de cinco segundos, dependendo da máquina).
- Uso de uma meta-linguagem como formato de dados para a comunicação, tornando-a mais lenta, mas que com a ajuda da boa compressão desse formato e da capacidade de auto-descrição do conteúdo, a verificação da boa formação e validade da mesma, se consegue ter mais ganhos do que perdas.
- A simplicidade de implementação de serviços. Embora depende muito dos objectivos dos serviços, da linguagem de implementação e das plataformas, estes podem ser implementados muito simplesmente usando as ferramentas correstes disponíveis.
- A rapidez do processamento. No exemplo, foi criado um serviço que acedia a uma base de dados MySQL para obter os dados a enviar ao cliente. Foi testado um pedido de informação de todos os alunos do sistema. Na base de dados existiam cerca de trezentos alunos e o processo demorou cerca de vinte segundos. Para pedidos com parâmetros de filtro para a resposta é muito mais rápida. Estes testes foram feitos estando o cliente e o serviço na mesma máquina. No entanto, a rapidez de resposta não está ao nível da comunicação, mas sim no tempo que o serviço demora a obter os dados do sistema de armazenamento e da preparação do documento de resposta.

### Desvantagens na utilização da metodologia

Esta metodologia pode não ser a mais adequada em organizações que dispõem de sistemas de informação distribuídos e com comunicações de dados automáticas já implementadas usando ferramentas específicas para o processo. Nestes casos o uso da metodologia pode complicar esse processo

e também diminuir o desempenho, nomeadamente ao nível do formato de dados da comunicação, que é uma meta-linguagem (XML), tornando a comunicação sempre mais lenta.

Esta metodologia também está limitada à comunicação de informação entre organismos, não proporcionando um ambiente de partilha de dados propriamente dito. Isso poderia ser feito ao nível dos serviços Web. No entanto a demonstração desta metodologia apenas foi feita através do uso de protótipos muito simples sem considerar questões de eficiência.

## 4.3 Perspectivas futuras

Existem alguns pontos da metologia apresentada que podem ser melhorados em trabalhos futuros, beneficiando da evolução das tecnologias envolvidas.

A especificação UML não permite criar relacionamentos entre classes de pacotes diferentes nos diagramas de classes. Isto obriga-nos a fazer um cópia das classes para outros pacotes sempre que se pretende fazer relacionamentos deste tipo. Um exemplo claro deste tipo de relacionamentos é a generalização da classe da `Disciplina` no pacote do organismo `Secretaria`. A possibilidade de relacionar classes de diferentes pacotes é uma possível inovação que não só traria vantagem ao nível da criação dos diagramas de classes, como também no processo de mapeamento para a obtenção das definições de dados *Schema*, sendo no entanto uma inovação não garantida.

No mapeamento que produz um *Schema* por cada organismo do modelo de dados seria interessante usar mais informação do modelo de dados. Já que os *Schemas* obtidos são usados para definir tipos de dados nos documentos WSDL, seria vantajoso gerar os próprios documentos WSDL como parte do processo de mapeamento. Neste trabalho foi definido um serviço exemplificativo do funcionamento da metodologia, contudo, para passarmos a definir o serviço no modelo de dados teríamos que especializar ainda mais os diagramas de classes UML. Outra hipótese é a utilização de outro tipo de diagramas UML como os diagramas de utilização definindo o comportamento dos organismos ao nível das funcionalidades que disponibilizam no seu serviço e que dados podem eles obter dos outros organismos.

## 4.4 Opinião pessoal

Desde que me foi proposto o tema deste projecto que fiquei bastante motivado visto ser uma problemática actual e pela qual tenho bastante interesse.

A metodologia foi sendo delineada de forma progressiva tendo sempre presente a ideia de usar *standards*, o que me obrigou nas várias etapas a inteirar-me das melhores opções para o trabalho. É claro que grande parte das tecnologias utilizadas eram-me desconhecidas, mas também foi uma forma de alargar os meus conhecimentos, apesar das muitas tecnologias estudadas não terem sido integradas na metodologia proposta.

Deste trabalho posso concluir que foi bastante produtivo, conseguindo atingir os objectivos que me levaram a frequentar este curso.

# Apêndice A

# Anexos

## A.1 WSDL

O documento WSDL seguinte define o serviço Web que foi usado no projecto de tese.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions
3      name="DataExchange"
4      targetNamespace="http://www.fc.up.pt/DataExchange"
5      xmlns="http://schemas.xmlsoap.org/wsdl/"
6      xmlns:data="http://www.fc.up.pt/Data-types"
7      xmlns:meta="http://www.fc.up.pt/Meta"
8      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
9      xmlns:tns="http://www.fc.up.pt/DataExchange"
10     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
11     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12     <xsd:import namespace="http://www.fc.up.pt/Data-types"
13                 schemaLocation="http://depa.fc.up.pt:8080/tese/data_definition.xsd
                       "/>
14     <xsd:import location="http://depa.fc.up.pt:8080/tese/meta_data.xsd"
15             namespace="http://www.fc.up.pt/Meta"/>
16     <message name="Request"/>
17
18     <message name="Reply">
19         <part name="Reply" type="data:SetDataObject"/>
20     </message>
21     <portType name="DataExchange">
22         <operation name="queryData">
23         <input message="tns:Request"/>
24             <output message="tns:Reply"/>
25         </operation>
26         <operation name="insertedData">
27         <input message="tns:Request"/>
28             <output message="tns:Reply"/>
29         </operation>
30         <operation name="updatedData">
31         <input message="tns:Request"/>
32             <output message="tns:Reply"/>
33         </operation>
34         <operation name="deletedData">
35         <input message="tns:Request"/>
36             <output message="tns:Reply"/>
37         </operation>
38     </portType>
39
```

```
40      <wsdl:binding name="DataExchangeBinding" type="tns:DataExchange">
41          <soap:binding style="rpc"
42                        transport="http://schemas.xmlsoap.org/soap/http"/>
43          <wsdl:operation name="queryData">
44            <soap:operation soapAction="queryData"/>
45            <wsdl:input>
46               <soap:body use="encoded"
47                          namespace="urn:xmltoday-delayed-quotes"
48                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
49               <soap:header element="meta:Request"/>
50            </wsdl:input>
51            <wsdl:output>
52               <soap:body use="encoded"
53                          namespace="urn:xmltoday-delayed-quotes"
54                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
55               <soap:header element="meta:Request"/>
56            </wsdl:output>
57          </wsdl:operation>
58          <wsdl:operation name="insertedData">
59            <soap:operation soapAction="insertedData"/>
60            <wsdl:input>
61               <soap:body use="encoded"
62                          namespace="urn:xmltoday-delayed-quotes"
63                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
64               <soap:header element="meta:Request"/>
65            </wsdl:input>
66            <wsdl:output>
67               <soap:body use="encoded"
68                          namespace="urn:xmltoday-delayed-quotes"
69                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
70               <soap:header element="meta:Request"/>
71            </wsdl:output>
72          </wsdl:operation>
73          <wsdl:operation name="updatedData">
74            <soap:operation soapAction="updatedData"/>
75            <wsdl:input>
76               <soap:body use="encoded"
77                          namespace="urn:xmltoday-delayed-quotes"
78                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
79               <soap:header element="meta:Request"/>
80            </wsdl:input>
81            <wsdl:output>
82               <soap:body use="encoded"
83                          namespace="urn:xmltoday-delayed-quotes"
84                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
85               <soap:header element="meta:Request"/>
86            </wsdl:output>
87          </wsdl:operation>
88          <wsdl:operation name="deletedData">
89            <soap:operation soapAction="deletedData"/>
90            <wsdl:input>
91               <soap:body use="encoded"
92                          namespace="urn:xmltoday-delayed-quotes"
93                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
94               <soap:header element="meta:Request"/>
95            </wsdl:input>
96            <wsdl:output>
97               <soap:body use="encoded"
98                          namespace="urn:xmltoday-delayed-quotes"
99                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
100              <soap:header element="meta:Request"/>
101           </wsdl:output>
102         </wsdl:operation>
103     </wsdl:binding>
104
105     <service name="DataExchange_Service">
```

```
106         <port binding="tns:DataExchangeBinding" name="urn:DataExchange">
107             <soap:address location="http://depa.fc.up.pt:8080/axis/services/
                    DataExchange"/>
108         </port>
109     </service>
110 </definitions>
```

Programa A.1: WSDL do serviço implementado.

## A.2 Aplicações

### A.2.1 Servidor

Antes de ser invocado o método pretendido, o *Axis* verifica se existe um método chamado `getHeaderInfo` e executa-o. Esse método foi definido e permite obter a meta-informação enviada no cabeçalho da mensagem SOAP pedido. Em seguida é executado o método (operação) que foi pedido[1]. Depois, é invocado o método `buildReply` que constrói a mensagem de resposta. A informação é obtida a partir de uma base de dados relacional.

```
1     public org.w3c.dom.Element queryData() throws java.rmi.RemoteException {
2         // Get the SOAPHeader Request Info
3         if (!getHeaderInfo())
4             return null;
5         try {
6             Node node;
7             if (R_query.length()>0) {
8                 org.w3c.dom.Document doc = (org.w3c.dom.Document)buildReply("
                      queryData");
9                 node = doQuery(doc);
10            } else {
11                node = buildReply("queryData");
12            }
13            return(createRootElement(node));
14        } catch (Exception e){
15            System.out.println("ERRO:  "+e);
16        }
17        return null;
18    }
19    public org.w3c.dom.Element insertedData() throws java.rmi.RemoteException {
20        // Get the SOAPHeader Request Info
21        if (!getHeaderInfo())
22            return null;
23        try {
24            Node node = buildReply("insertedData");
25            return(createRootElement(node));
26        } catch (Exception e){
27            System.out.println("ERRO:  "+e);
28        }
29        return null;
30    }
31    public org.w3c.dom.Element updatedData() throws java.rmi.RemoteException {
32        // Get the SOAPHeader Request Info
33        if (!getHeaderInfo())
34            return null;
35        try {
36            Node node = buildReply("updatedData");
37            return(createRootElement(node));
38        } catch (Exception e){
```

[1] O método `deletedData` não foi implementado.

```
39              System.out.println("ERRO:  "+e);
40          }
41          return null;
42      }
43      public org.w3c.dom.Element deletedData() throws java.rmi.RemoteException {
44          return null;
45      }
```

Programa A.2: Aplicação Java do serviço.

O método `getHeaderInfo` passa a meta-informação do cabeçalho da mensagem SOAP pedido para as variáveis estáticas da classe.

```
1       public boolean getHeaderInfo() throws AxisFault {
2           MessageContext context = MessageContext.getCurrentContext();
3           Message msg = context.getRequestMessage();
4           SOAPEnvelope env = msg.getSOAPEnvelope();
5
6           SOAPHeaderElement header = env.getHeaderByName("http://www.fc.up.pt/Meta",
7                                                          "Request");
8           if (header != null) {
9                   try {
10                      // Get Attribute values
11                      R_source = header.getAttributeValue("source");
12                      R_ID = header.getAttributeValue("ID");
13                      R_timestamp = header.getAttributeValue("timestamp");
14                      R_object = header.getAttributeValue("object");
15                      R_query = header.getAttributeValue("query");
16                      R_sinceDate = header.getAttributeValue("sinceDate");
17                  } catch (Exception e) {
18                      throw AxisFault.makeFault(e);
19                  }
20          } else
21              return false;
22
23          return true;
24      }
```

Programa A.3: Meta-informação no cabeçalho das mensagens SOAP.

O método `buildReply` cria um documento XML com os dados pretendidos. Dependendo do tipo de pedido de informação que foi efectuado, executa uma instrução SQL sobre a base de dados, neste caso MySQL. Neste caso não existe um processo de mapeamento entre a estrutura de dados do sistema e a estrutura de um documento XML, já que a estrutura da base de dados foi criada a partir do *Schema* do sistema[2]. A cada registo equivale um elemento XML relativo a um objecto, e para cada propriedade do registo é adicionado um novo elemento ao elemento objecto. Este é o formato definido pelo *Schema*.

```
1       public Node buildReply(String operation) {
2           int j;
3
4           Driver name;
5           Enumeration enum;
6           BufferedReader input = new BufferedReader (new InputStreamReader(System.in
                ));
7
8           try {
9               // Document Builder
10              DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

[2]Esta estrutura foi definida a partir do *Schema* do sistema, através da aplicação da folha de estilos para gerar a informação no formato SQL.

```
11              DocumentBuilder docb = dbf.newDocumentBuilder();
12              org.w3c.dom.Document doc = docb.newDocument();
13
14              Element root,obj,item;
15
16              // Root Element
17              root = doc.createElement("Document");
18
19              // Get Info from Database
20              Class.forName("org.gjt.mm.mysql.Driver");
21              Connection db = DriverManager.getConnection("jdbc:mysql:///tese?user=
                    rogerio");
22              Statement st = db.createStatement();
23
24              String temp = "SELECT * FROM "+R_object;
25
26              if (operation.equals("queryData")) {
27                  if (R_sinceDate.length()>0)
28                      temp = temp+QUERY_STATEMENT+" '"+R_sinceDate+"'";
29              } else if (operation.equals("insertedData")) {
30                  if (R_sinceDate.length()>0)
31                      temp = temp+INSERTED_STATEMENT+" '"+R_sinceDate+"'";
32              } else if (operation.equals("updatedData")) {
33                  if (R_sinceDate.length()>0)
34                      temp = temp+UPDATED_STATEMENT+" AND updateDate >= '"+
                            R_sinceDate+"'";
35                  else temp = temp+UPDATED_STATEMENT+" '"+R_sinceDate+"'";
36              }
37
38              ResultSet rs = st.executeQuery(temp);
39              ResultSetMetaData md = rs.getMetaData();
40
41              while(rs.next()) {
42                  // Object Element
43                  obj = doc.createElement("data:"+R_object);
44
45                  for(j=1;j<=md.getColumnCount();j++){
46                      // For each columns add a new Element
47                      if (md.getColumnName(j).equals("insertDate") || md.
                            getColumnName(j).equals("updateDate"))
48                          continue;
49                      String data = "data:"+md.getColumnName(j);
50                      item = doc.createElement(data);
51                      item.appendChild(doc.createTextNode(rs.getString(j)));
52                      obj.appendChild(item);
53                  }
54                  // Add Object Element to Root
55                  root.appendChild(obj);
56              }
57
58              doc.appendChild(root);
59              rs.close();
60              st.close();
61              db.close();
62              return(doc);
63          } catch (Exception e) {
64              System.out.println("Erro... "+e);
65          }
66          return null;
67      }
```

Programa A.4: Metodo que constrói o documento de resposta.

Caso tenha sido enviada no pedido uma expressão XPath, esta é aplicada no método `doQuery`, sobre o documento com todos os objectos de uma classe, segundo as restrições do método e da data.

```
1     public Node doQuery(org.w3c.dom.Document doc) {
2     try{
3         // Query the Document
4         Transformer serializer = TransformerFactory.newInstance().newTransformer();
5         serializer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
6
7         System.out.println("QUERY:     /Document"+R_query);
8         NodeIterator nl = XPathAPI.selectNodeIterator(doc, "/Document"+R_query);
9
10        // Get the Result Nodes
11        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
12            DocumentBuilder docb = dbf.newDocumentBuilder();
13            org.w3c.dom.Document rdoc = docb.newDocument();
14        org.w3c.dom.Element topo = rdoc.createElement("Document");
15        Node n,n2;
16        DOMResult d = new DOMResult();
17        while ((n = nl.nextNode())!= null)
18        {
19            n2 = rdoc.importNode(n,true);
20            topo.appendChild(n2);
21        }
22        rdoc.appendChild(topo);
23        return(rdoc);
24    } catch (Exception e) {
25        System.out.println("Erro... "+e);
26    }
27    return null;
28    }
```

Programa A.5: Aplicação do filtro sobre o documento de objectos.

O retorno do método do serviço invocado tem de corresponder a apenas um elemento[3]. Como o documento obtido pode ter vários, é necessário reunir esses elementos dentro de um outro que os agregue. Por outro lado, o documento obtido não está de acordo com o *Schema*: tem de existir o elemento `SetDataObject` que incorpora todos os outros elementos. Portanto é necessário invocar o método `createRootElement` que faça essa tarefa, e só depois retornar esse elemento.

```
1     public Element createRootElement(Node node) {
2         try {
3             org.w3c.dom.Document doc2 = (org.w3c.dom.Document) node;
4             Element root2 = doc2.getDocumentElement();
5
6             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
7             DocumentBuilder docb = dbf.newDocumentBuilder();
8             org.w3c.dom.Document doc = docb.newDocument();
9
10            Element root = doc.createElement("data:SetDataObject");
11            root.setAttribute("xmlns:data","http://www.fc.up.pt/Data-types");
12            root.setAttribute("xmlns:xsi","http://www.w3.org/2001/XMLSchema-
                  instance");
13            root.setAttribute("xsi:schemaLocation","http://www.fc.up.pt/Data-types
                  http://depa.fc.up.pt:8080/tese/data_definition.xsd");
14            root.appendChild(doc.createTextNode("\n"));
15
16            // Import Object Elements
17            for(Node child = root2.getFirstChild(); child != null; child = child.
                  getNextSibling())
18            {
19                int type = child.getNodeType();
20                if (type==Node.ELEMENT_NODE) {
21                    Node imp= doc.importNode(child,true);
22                    root.appendChild(imp);
```

[3]Esse elemento é que compõem o corpo da mensagem SOAP

```
23                  }
24              }
25          return(root);
26      }catch (Exception e){
27      }
28      return null;
29  }
```

Programa A.6: Cria o elemento de raiz do documento de resposta.

Quando a aplicação retorna o documento resultante, que equivale ao corpo da mensagem SOAP, tem de definir alguma meta-informação de retorno ao cliente. Para isso definiu-se um método com o nome `setHeaderInfo` que é invocado por uma classe definida pelo *Axis*, antes da mensagem ser enviada.

```
1   public void setHeaderInfo() throws AxisFault {
2       MessageContext context = MessageContext.getCurrentContext();
3       Message msg = context.getResponseMessage();
4       SOAPEnvelope env = msg.getSOAPEnvelope();
5
6       try {
7           DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
8           DocumentBuilder docb = dbf.newDocumentBuilder();
9           org.w3c.dom.Document doc = docb.newDocument();
10
11          java.util.Date date = new java.util.Date();
12          SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");
13
14          String timestamp = df.format(date);
15
16          org.w3c.dom.Element reply = doc.createElement("meta:Reply");
17          reply.setAttribute("xmlns:meta","http://www.fc.up.pt/Meta");
18          reply.setAttribute("source","DCC");
19          reply.setAttribute("ID","id999");
20          reply.setAttribute("requestID",R_ID);
21          reply.setAttribute("SOAP-ENV:mustUnderstand","1");
22          reply.appendChild(doc.createTextNode("\n"));
23
24          SOAPHeaderElement header = new SOAPHeaderElement(reply);
25          env.addHeader(header);
26          //              System.out.println(env);
27      }catch (Exception e) {
28
29      }
30  }
```

Programa A.7: Meta-informação no cabeçalho da resposta.

### A.2.2 Cliente

A aplicação cliente (`Client.java`) está organizada em várias secções. Existem algumas pouco relevantes, como as definições estáticas no início da classe e o método `walk`, que permite fazer o *output* de um documento XML.

O primeiro método relevante que a aplicação executa é o `doMain`. Este tem como função identificar todos os parâmetros de execução e definir os valores das variáveis estáticas da classe. Também faz algumas validações em relação aos parâmetros passados.

```
1   public void doMain(String [] args,String name){
2       // Verify the command line options
3       try {
```

```
4           int i = 0;
5           while (true) {
6               if (i>args.length) badUsage(name, "No arguments");
7               if (i==args.length) break;
8
9               if (args[i].charAt(0)=='-') {
10
11                  if (args[i].equals("-m")) {
12                      i++;
13                      setMethodName(args[i++]);
14                  }
15                  else if (args[i].equals("-O")) {
16                      i++;
17                      setObject(args[i++]);
18                  }
19                  else if (args[i].equals("-q")) {
20                      i++;
21                      setQuery(args[i++]);
22                  }
23                  else if (args[i].equals("-d")) {
24                      i++;
25                      setSinceDate(args[i++]);
26                  }
27                  else if (args[i].equals("-lx")) {
28                      setLogXML(true);
29                      i++;
30                  }
31                  else if (args[i].equals("-lh")) {
32                      setLogHTML(true);
33                      i++;
34                  }
35                  else if (args[i].equals("-lh")) {
36                      setLogSQL(true);
37                      i++;
38                  }
39
40                  else badUsage(name, "Unknown option " + args[i]);
41              }
42
43              else break;
44          }
45          if (i==0) badUsage(name, "No arguments");
46          if (getMethodName()==null)
47              quit("No Method Specified",1);
48          if (getObject()==null)
49              quit("No Object Specified",1);
50          if (!getMethodName().equals("queryData") &&
51          !getMethodName().equals("insertedData") &&
52              !getMethodName().equals("updatedData") && !getMethodName().equals("
                    deletedData"))
53              badUsage(name, "Invalid Method");
54          if (getQuery()!=null && !getMethodName().equals("queryData"))
55              System.out.println("INFO: The query only have efect in method \"
                    queryData\"!");
56          if (getLogSQL() && !getMethodName().equals("insertedData") &&
57              !getMethodName().equals("updatedData") && !getMethodName().equals("
                    deletedData"))
58
59      } catch (Exception e) {
60          System.err.println("error:  "+e.getMessage());
61          e.printStackTrace();
62      }
63      doCall();
64      System.exit(0);
```

```
65      }
```

Programa A.8: Método que trata dos parâmetros passados na linha de comandos.

Em seguida o pedido começa a ser processado pelo método `doCall`. Este cria a mensagem SOAP. Começa por invocar o método `setServiceDefault` para definir informação relativa ao serviço. Esta informação depende do serviço que se vai invocar. Depois adiciona a meta-informação do pedido ao cabeçalho da mensagem. Em seguida faz o pedido ao serviço e fica à espera da resposta. Depois de obtida a resposta do serviço, como um objecto da classe `Element`, é invocado o método `parseReply`.

```
1       public static void setServiceDefault(){
2           //setServiceName("http://depa.fc.up.pt:7080/DataExchange");
3           setServiceName("urn:DataExchange");
4           setServiceURL("http://depa.fc.up.pt:8080/axis/servlet/AxisServlet");
5           setSource("FCUP");
6           setXSLFile("xmltohtml.xsl");
7       }
8       public static void doCall(){
9           // Build a timestamp with date/time
10          java.util.Date date = new java.util.Date();
11          SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");
12          timestamp = df.format(date);
13          // Set some properties with default values
14          setServiceDefault();
15          try {
16              // Build a service call
17              ServiceCall sc = new ServiceCall();
18              // Set properties of service call
19              sc.begin();
20              sc.setServiceQN(getServiceName(),getMethodName());
21              sc.setServiceURL(getServiceURL());
22              // Build the request element
23              Element request = buildRequest();
24              SOAPHeaderElement header = new SOAPHeaderElement(request);
25              // Add the element to the soap header
26              sc.addHeader(header);
27              // Invoke the service
28              Element result = sc.invoke();
29              // Parse the result document
30              parseReply(result);
31          } catch (Exception e) {
32              quit("Call Service Error",1);
33              System.out.println("Error : " + e.toString());
34          }
35      }
```

Programa A.9: Define o serviço por defeito.

O método `parseReply` cria um documento XML a partir do objecto da classe `Element` obtido na resposta. Em seguida faz uma validação sintáctica do documento relativamente ao *Schema*. Por fim invoca os métodos que permitem guardar o documento no formato pretendido.

```
1       public static void parseReply(Element result) {
2           try {
3               // Add the prototype information
4               String document = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"+
                    XMLUtils.ElementToString(result);
5               System.out.println(document);
6
7               // Create the Reader
8               XMLReader reader = XMLReaderFactory.createXMLReader("org.apache.xerces.
                    parsers.SAXParser");
9               // Set Features
```

```
10              reader.setFeature("http://xml.org/sax/features/validation", true);
11              reader.setFeature("http://apache.org/xml/features/validation/schema",
                    true);
12              reader.setFeature("http://apache.org/xml/features/validation/schema-
                    full-checking", true);
13              reader.setContentHandler(new DefaultHandler());
14              // Parse
15              try {
16                  reader.parse(new InputSource(new StringReader(document)));
17                  // Save result as XML and|or HTML
18                  if (getLogXML())
19                      logXMLFile(document);
20                  if (getLogHTML())
21                      logHTMLFile(document);
22                  if (getLogSQL())
23                      logSQLFile(document);
24              }
25              catch (SAXParseException e) {
26                  System.err.println("error: SAXParser Error "+e.getMessage());
27              }
28              catch (Exception e) {
29                  System.err.println("error: Parse error occurred - "+e.getMessage())
                        ;
30                  if (e instanceof SAXException) {
31                      e = ((SAXException)e).getException();
32                  }
33                  e.printStackTrace(System.err);
34              }
35          } catch(Exception e){
36              System.err.println("error: Client.parseReply "+e.getMessage());
37          }
38      }
```

Programa A.10: Método que verifica o documento de resposta.

O método `logXMLFile` grava o documento XML num ficheiro. O método `logHTMLFile` aplica uma folha de estilos XSLT sobre o documento XML obtendo a informação em HTML, também gravada gravada num ficheiro. O método `logSQLFile` grava o ficheiro como instruções SQL da mesma forma que o anterior, ou sejam, aplica uma folha de estilos XSLT sobre o documento XML. O nome-caminho destes ficheiros é definido pela seguinte sequência: nome do serviço, seguido do método do pedido e do *timestamp*(data e hora) do pedido.

```
1       // Save result document as SQL file
2       public static void logXMLFile(String document){
3           String filename = ""+getServiceName()+"_"+getMethodName()+"_"+timestamp+".
                xml";
4           PrintWriter filew;
5           String topInfo = "<!-- Service: "+getServiceName()+"\n Method: "+
                getMethodName()+"\n Object: "+getObject();
6           if (getQuery()!=null)
7               topInfo = topInfo+"\n Query: "+getQuery();
8           topInfo = topInfo+"\n Timestamp: "+timestamp+" -->";
9           try {
10              File file = new File(filename);
11              if (!file.exists())
12                  file.createNewFile();
13              filew = new PrintWriter(new FileWriter(filename),true);
14              filew.println(topInfo);
15              filew.println(document);
16          } catch (IOException e) {
17              System.out.println("FILE ERROR: "+e);
18              filew = new PrintWriter(System.out,true);
19              filew.println(document);
20          }
```

```
21      }
22      // Save result document as HTML file
23      public static void logHTMLFile(String document){
24          String filename = ""+getServiceName()+"_"+getMethodName()+"_"+timestamp+".
                html";
25          try {
26              TransformerFactory tfactory = TransformerFactory.newInstance();
27              Transformer transformer = tfactory.newTransformer(new StreamSource(
                    getXSLFile()));
28
29              Reader       xmlReader = new StringReader(document);
30              StreamSource xmlSource = new StreamSource(xmlReader);
31
32          transformer.transform(xmlSource,
33                  new StreamResult(new PrintWriter(new FileWriter(new File(
                        filename)))));
34
35      } catch(TransformerConfigurationException tce) {
36          System.out.println("ERROR: "+tce);
37      } catch(TransformerException te) {
38          System.out.println("ERROR: "+te);
39      } catch(IOException ioe) {
40          System.out.println("ERROR: "+ioe);
41      }
42      }
43      // Save result document as SQL file
44      public static void logSQLFile(String document){
45          String filename = ""+getServiceName()+"_"+getMethodName()+"_"+timestamp+".
                sql";
46          String stylefile;
47          try {
48              TransformerFactory tfactory = TransformerFactory.newInstance();
49              stylefile = "xmltosql_insert.xsl";
50              if (getMethodName().equals("updatedData"))
51                  stylefile = "xmltosql_update.xsl";
52              else if (getMethodName().equals("deletedData"))
53                  stylefile = "xmltosql_delete.xsl";
54              Transformer transformer = tfactory.newTransformer(new StreamSource(
                    stylefile));
55
56              Reader       xmlReader = new StringReader(document);
57              StreamSource xmlSource = new StreamSource(xmlReader);
58
59              transformer.transform(xmlSource,
60                                    new StreamResult(new File(filename)));
61
62          } catch(TransformerConfigurationException tce) {
63              System.out.println("ERROR: "+tce);
64          } catch(TransformerException te) {
65              System.out.println("ERROR: "+te);
66          }
67
68      }
```

Programa A.11: Método que grava o resultado em ficheiros.

# Referências


[Alh98] Sinan Si Alhir. *UML in a nutshell - A Desktop Quick Reference.* O'Reilly, 1st edition, 1998.

[Amb00] Scott W. Ambler. Mapping Objects to Relational Databases. RONIN International White Papers, http://www.ronin-intl.com, June 2000.

[ANSI] American National Standards Institute. SQL: Structured Query Language. ANSI X3.135/1992, http://www.iso.ch.

[Aps02] Kapil Apshankar. Ws-security, security for web services. http://www.webservicearchitect.com, 2002.

[AXI] AXIS. The Apache XML Project, http://xml.apache.org/.

[Bea01] Mark Birbek and et al. *Professional XML.* Wrox Publisher, 2nd edition, 2001.

[Bou] Ronald Bourret. XML-DBMS, Version 1.0.1. http://www.rpbourret.com.

[BRJ99] Grady Booch, James Runbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1st edition, 1999.

[Bro99] S. Brodky. XMI Opens Applications Interchange. IBM Articles, http://www.ibm.com, July 1999.

[COR] CORBA: Common Object Request Broker Architecture. Object Management Group, http://www.omg.org.

[dBW] Bas de Bakker and Irsan Widarto. An Introduction to XQuery. http://perfectxml.com/articles/xml/xquey.asp.

[Dea02] H. M. Deitel and et al. *XML How to Program.* Prentice Hall, 1st edition, 2002.

[dJea02] Irmen de Jong and et al. Web Services/SOAP and CORBA, 2002.

[DOM] DOM: Document Object Model. World Wide Web Consortium, http://www.w3.org/.

[Fes01] Stephanie Fesler. Using DOM to Traverse XML. O'Reilly OnJava.com, http://www.onjava.com, August 2001.

[For99] Paul J. Fortier. *SQL-3 Implementing the Object-Relational Database.* McGraw-Hill, 1st edition, 1999.

[Gis01] Dan Gisolfi. Is Web services the reincarnation of CORBA? IBM developerWorks, http://www.ibm.com, July 2001.

[HH01] Larry S. Hayashi and John Hatton. Combining UML, XML and relational database technologies - the best of all worlds for robust linguistic databases. Technical report, SIL International, http://www.sil.org, 2001.

[Hou01] Lori Houston. SOAP Security Issues. Sun Developers Notebook, http://www.sun.com, December 2001.

[HTT] HTTP: Hypertext Transfer Protocol. World Wide Web Consortium, http://www.w3.org/.

[Ins99] MageLang Institute. Introduction to CORBA. Short Course. http://java.sun.com, December 1999.

[Kum99] S. Ilango Kumaran. The Specification for the Metadata Interchange Format. *XML Developers Journal*, 1999.

[Lan02] Eoin Lane. Is WSDL the indispensable API? JavaWorld, http://www.javaworld.com, May 2002.

[Len] Evan Lenz. XQuery: Reinventing the Whell? http://www.xmlportfolio.com/xquey.html.

[MG98] José Alves Marques and Paulo Guedes. *Tecnologia de Sistemas Distribuídos.* Prentice Hall, 1st edition, 1998.

[MIM] MIME: Multipurpose Internet Mail Extensions. World Wide Web Consortium, http://www.w3.org/.

[New02] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI.* Addison Wesley Professional, 1st edition, 2002.

[ODL] ODL: Object Data Language. Object Data Management Group, http://www.odmg.org.

[OH97] Robert Orfali and Dan Harkey. *Client/Server Programming with JAVA and CORBA.* Prentice Hall, 1st edition, 1997.

[OLE] OLE: Object Linking Embedding. Microsoft, http://www.microsoft.com.

[Ols] Mike Olson. Introduction to CORBA. part 1. http://www.linuxworld.com.

[oPS02] University of Puget Sound. Introducing XHTML. April 2002.

[PM] Warwick Ford Phillip M., Hallan-Baker. XML Key Management Specification (XKMS).

[RG] Raghu Ramakrishnan and Johannes Gehrke. *Database Managment Systems.* 2.

[SAX] SAX: Simple API for XML. http://www.saxproject.org/.

[Sch96] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc, 2nd edition, 1996.

[SER] SERVLET. Java Comunity Process, http://jcp.org.

[Sko99] David Skogan. UML as a Schema Language for XML based Data Interchange. In *UML'99 - The Unified Modeling Language Beyond the Standard*, 1999.

[SOA] SOAP: Simple Object Access Protocol. World Wide Web Consortium, http://www.w3.org/.

[SR01] Len Seligman and Arnon Rosenthal. XML's Impact on Database and Data Scharing. 2001.

[Sun01] Todd Sundsted. Building Security into Web Services. Sun Developers Notebook, http://www.sun.com, August 2001.

[TGBS99] Zar Zar Tun, Andrew Goodchild, Linda Bird, and Hoylen Sue. Introduction to XML Schema. Technical report, DSTC, The University of Queensland, Australia, October 1999.

[Tid00] Doug Tidwell. Web Services - The Web's next revolution. IBM Tutorials, http://www.ibm.com, November 2000.

[TOM] TOMCAT. The Apache Jakarta Project, http://jakarta.apache.org/.

[Tur99] Volker Turau. Making legacy data accessible for XML applications. http://www.informatik.fh-wiesbaden.de, 1999.

[Vas00] Clemens Vasters. Why SOAP doesn't lack security while it does. http://www.newtelligence.com, 2000.

[Vas01] Venu Vasudevan. A Web Services Primer. O'Reilly XML.com, http://www.xml.com, April 2001.

[Wil01] Kevin Williams. Why XML Schema beats DTDs hands-down for data. IBM developerWorks, http://www.ibm.com, June 2001.

[WSD] WSDL: Web Services Description Language. World Wide Web Consortium, http://www.w3.org/.

[XMI] XMI: XML Metadata Interchange. Object Management Group, http://www.omg.org.