UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

XML-ODBMS: Um Ferramenta de Intercâmbio de Dados entre Bancos de Dados Orientados a Objeto

Nilton Oliveira Matos Júnior

CAMPINA GRANDE – PB

Fevereiro / 2003

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

XML-ODBMS: Uma Ferramenta de Intercâmbio de Dados entre Bancos de Dados Orientados a Objetos

Nilton Oliveira Matos Júnior

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

Orientador: Marcus Costa Sampaio

Campina Grande, Paraíba, Brasil
Fevereiro de 2003

FICHA CATALOGRÁFICA

MATOS Jr, Nilton Oliveira

M425F

XML-ODBMS: Uma Ferramenta de Intercâmbio de Dados entre Bancos de Dados Orientados a Objetos e Objeto-Relacionais

Dissertação de Mestrado, Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Fevereiro de 2003.

164p.

Orientador: Marcus Costa Sampaio

Palavras Chave:

1. Banco de Dados
2. Intercâmbio de Dados
3. XML

CDU - 681.3.07B

XML-ODBMS: Uma Ferramenta de Intercâmbio de Dados entre Bancos de Dados Orientados a Objeto

Nilton Oliveira Matos Júnior

Dissertação aprovada em: 27 de fevereiro de 2003

Marcus Costa Sampaio

Campina Grande, 27 de fevereiro de 2003

Agradecimentos

Aos meus pais, Nilton Oliveira Matos e Iva Margariada Montes Vieira Matos, e aos meus irmãos Carla e Lucas, pelo apoio dispensado durante o tempo de desenvolvimento do trabalho.

Ao meu orientador, Marcus Costa Sampaio, por sua dedicação e fundamental ajuda à conclusão deste trabalho.

A todos os professores do Mestrado que me concederam conhecimento suficiente para encarar este desafio.

Resumo

Este trabalho apresenta uma ferramenta capaz de realizar o intercâmbio de dados entre bancos de dados orientados a objetos (BDOO), incluindo também os bancos de dados objeto-relacionais (BDOR). Uma operação de intercâmbio consiste na exportação/importação de dados entre banco de dados, utilizando XML como linguagem intermediária, ou linguagem de intercâmbio. Os intercâmbios são feitos em um alto grau de abstração e interoperabilidade, através de uma API Java ou de uma *interface* gráfica. Nosso trabalho vem preencher uma lacuna existente na literatura: do nosso conhecimento, todo o esforço de pesquisa em interoperabilidade de bancos de dados tem se concentrado somente em bancos de dados relacionais.

Abstract

This work presents a tool capable to exchange data between object-oriented databases(DBOO) and object-relational databases(DBOR). An exchanging operation consists on export/import of data from data between databases through XML as an intermidiate language or exchange language. The exchange data are done through a high level of abstraction and they are also non-stopping throughout Java API or graphic interface. Our work has the intention to fill in a gaap existent in this kind of literature: from our knowledge, all of research efforts on non-stopping database exchange have been concentrated on relational database.

Lista de Tabelas

Lista de Figuras

Sumário

Capítulo 1

Introdução

1.1 Contexto da Pesquisa

A crescente evolução da Internet (Web), no que diz respeito à quantidade crescente de pessoas que desfrutam de todos os serviços e informações disponíveis nesta verdadeira rede de integração mundial, fez com empresas começassem a explorar e a investir maciçamente neste forte mercado consumidor. Para tanto, muitas empresas desenvolvedoras de *software* começaram a criar soluções que possibilitam a extensão da comercialização de produtos e serviços, saindo de uma simples venda no balcão ou por telefone para um verdadeiro comércio virtual.

Além da comercialização de produtos e serviços, as empresas viram que a Web poderia ser utilizada, não apenas como mais uma forma de vender seus produtos e serviços, mas também como uma maneira de se comunicar com seus parceiros comercias (operações *Businnes-to-Businnes*), compartilhando informações com o objetivo de obter benefícios comerciais. Segundo [SUN1 00], as operações *Businnes-to-Businnes* (B2B) chegarão a movimentar em 2004 cerca de 2,7 trilhões de dólares. Em pesquisa realizada pela *NerveWire*[1] em 2002, verificou-se que uma grande porcentagem das empresas entrevistadas aperfeiçoaram seus produtos e serviços por meio do compartilhamento de informações com seus parceiros comerciais.

O grande desafio a ser superado pelos desenvolvedores de soluções B2B consiste em construir aplicações de intercâmbio de informações que sejam independentes de plataforma, e que permita a visualização das informações, que estão sendo tratadas, de uma forma simples e clara, onde qualquer pessoa possa interpretá-las e compreendê-las sem a necessidade da utilização de sistemas específicos [SUN1 00]. Uma forma de permitir o intercâmbio de informações entre sistemas heterogêneos é adotar um formato padrão para saída de informações para os sistemas de exportação e um formato padrão para entrada de informações

[1] Empresa de consultoria de gerenciamento e de integração de sistemas

para os sistemas de importação. A linguagem XML possibilita a definição deste padrão [BOS 97].

XML, *eXtensible Markup Language*, é uma linguagem de marcação de texto, em que os marcadores (*tags*) podem ser criados de acordo com a necessidade. Esta flexibilidade possibilita que as informações contidas em um documento XML possam ser estruturadas e identificadas, a fim de permitir a leitura e a extração/interpretação dessas informações de forma fácil, simples e sobre qualquer plataforma. A flexibilidade e a transportabilidade desta linguagem são propriedades que a credenciam para a utilização em qualquer tipo de aplicação em que seja necessário o intercâmbio de dados entre sistemas heterogêneos. Antes do surgimento da XML, para o desenvolvimento de aplicações de intercâmbio precisava-se definir um formato particular para a estruturação das informações que iam ser compartilhadas, sendo então uma solução específica, com pouca ou nenhuma flexibilidade para adaptá-la a outros contextos de intercâmbio. Ao contrário, XML oferece muita flexibilidade, através da padronização dos marcadores entre uma comunidade de usuários que intercambiam informações [MOR 00, LOP 01].

Aplicações B2B são cenários típicos de intercâmbio entre informações diversas. Muitas vezes, faz-se necessário que aplicações diferentes (com finalidades e tecnologias diferentes) possam intercambiar dados a fim de obter resultados relevantes para a empresa. O intercâmbio de dados é uma forma de integração de aplicações, fazendo com que empresas possam se comunicar com parceiros, fornecedores e clientes, mesmo que estes estejam dispostos em ambientes de tecnologias diferentes [REN 02]. Como exemplo, podemos citar a necessidade das indústrias farmacêuticas em trocar informações de seus produtos com hospitais, clínicas, fornecedores e farmácias, com o objetivo de divulgar seus produtos e de obter um *feedback* sobre aceitação do mercado.

Os dados que serão intercambiados geralmente encontram-se armazenados em alguma fonte de dados, como em banco de dados [MOR 00]. Para que as aplicações possam integrar-se, é necessário que os dados sejam estruturados em algum documento, de onde possam ser lidos, interpretados e extraídos. Como hoje a maioria das aplicações se serve de Sistemas de Gerência de Banco de Dados (SGBDs) para o armazenamento e a consulta aos dados, o intercâmbio de dados entre aplicações consiste muitas vezes da exportação dos dados de um SGBD para um documento estruturado, e na importação dos dados contidos no documento estruturado para um outro SGBD.

O grande desafio a ser superado é portanto, oferecer ferramentas capazes de realizar a importação/exportação dos dados armazenados em SGBDs, necessários para a integração de

aplicações, ainda oferecendo transportabilidade, simplicidade e flexibilidade, permitindo que os desenvolvedores de aplicação se preocupem unicamente com a lógica de negócio de sua aplicação.

1.2 Sistema de Intercâmbio de Dados Utilizando XML

A principal motivação para a construção de um Sistema de Intercâmbio de Dados entre Bancos de Dados (SIBD) advém da necessidade de construir sistemas de integração de aplicações (EAI – "**E**nterprise **A**pplication **I**ntegration") que possibilitem o intercâmbio de informações entre organizações [MOR 00].

SIBDs são *softwares* que exportam os dados de um determinado banco de dados sob um SGBD, estruturando-os em um arquivo intermediário, o qual é então importado para um outro banco de dados, provavelmente sob um outro SGBD [BOU 00]. A transportabilidade da linguagem XML e a sua capacidade de representar informações, colocam-na como a ferramenta ideal para a construção dos arquivos intermediários, em aplicações de intercâmbio de dados [WIL 00, SHA00]. Mais especificamente, XML aparece como a solução mais indicada como uma *interface* entre bancos de dados origem (BDO) e destino (BDD), em aplicações de intercâmbio de dados. Pode-se seguramente afirmar que a maioria dos SIBDs utilizam a linguagem XML (SIBD XML) para as operações de intercâmbio de dados entre SGBDs. A figura 1.1 é uma ilustração da arquitetura de um SIBD.

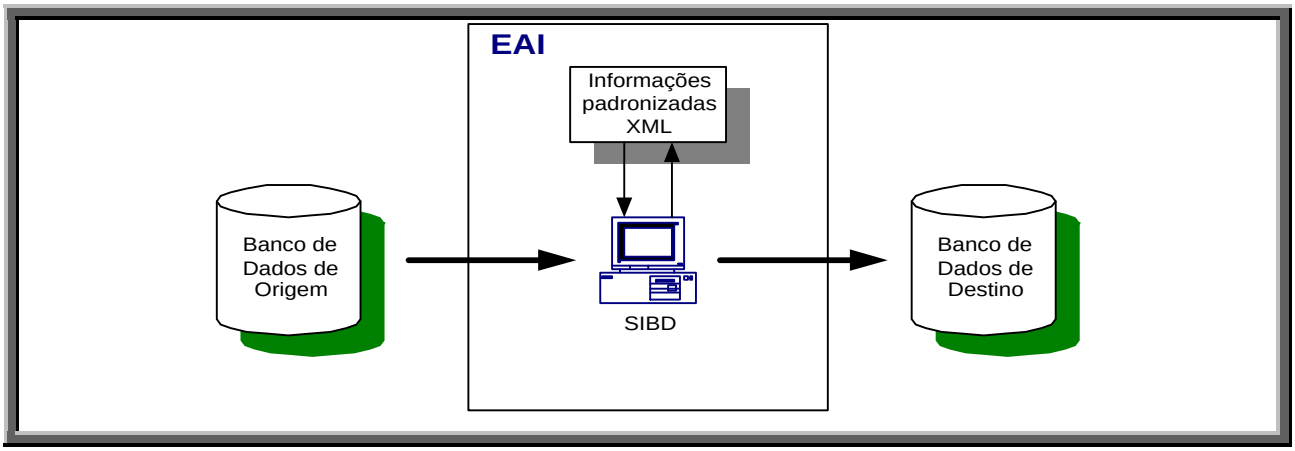

Figura 1.1: Esquema básico do funcionamento de um SIBD

Devido à diversidade tecnológica existente hoje, o número de soluções criadas sobre diferentes tecnologias vem aumentando, tornando mais difícil e complicada a atuação de

aplicações EAI. Para que um SIBD possa ser utilizado com sucesso em aplicações EAI, neste universo de sistemas heterogêneos, é necessário que ele possa ser utilizado em diferentes plataformas (*Windows*, *Unix*, *Linux*), que ofereça suporte a diferentes SGBDs (*Oracle*, *SQL Server*, *Informix*, *Interbase*, *DB2*, *Jasmine*) e aos modelos mais comuns de banco de dados tratados por esses SGBDs (modelo relacional - R, ou modelo orientado a objeto - OO, ou ainda o modelo objeto-relacional - OR), e que possibilite o intercâmbio de dados que se diferenciam pelos seus tipos e estruturas.

1.3 Objetivos e Relevância do Trabalho

O objetivo deste trabalho é desenvolver um SIBD XML, XML-ODBMS, que permitirá o intercâmbio de dados entre SGBDs OO/OR/R, oferecendo aos desenvolvedores de aplicações EAI um pacote de classes Java, que poderão ser incorporadas conforme suas necessidades, e uma *interface* gráfica que implemente estas classes, tudo com o objetivo de facilitar a tarefa de exportação/importação de dados, fundamental para aplicações EAI com SIBDs.

A relevância do XML-ODBMS reside no fato que ele contempla também os BDOOs e os BDORs, o que não acontece com os SIBDs existentes. Desta forma, XML-ODBMS atenderá igualmente às organizações com as mais recentes tecnologias de sistemas de informação, como os modernos SGBDOOs e SGBDORs.

1.4 Organização da Dissertação

No capítulo 2, fazemos uma apresentação da linguagem XML, mostrando seus principais elementos, aplicações, como ela pode ser utilizada em operações de intercâmbio de dados, e como ela pode ser manipulada através da tecnologia Java .

No terceiro capítulo, mostramos as principais características de um SIBD, com destaque para a análise comparativa de alguns SIBDs XML.

Os três capítulos seguintes são o cerne de nossa dissertação. O capítulo 4 é dedicado à discussão da linguagem XML-ODMG, desenvolvida para descrever dados armazenados tanto em SGBDs OO quanto em SGBDs OR, utilizada como a linguagem intermediária para exportação/importação de dados através do XML-ODBMS. No capítulos 5 destacamos o

XML-ODBMS, mostrando suas características, funcionalidades, arquitetura, componentes e detalhes da implementação.

Já no capítulo 6 demonstramos a utilização do XML-ODBMS através da implementação de um estudo de caso.

No capítulo 7, apresentamos as conclusões e as perspectivas do nosso trabalho.

Capítulo 2

XML como Padrão de Representação de Informações

2.1 Introdução

Este capítulo explana sobre a linguagem XML ("e**X**tensible **M**arkup **L**anguage") [XML 02] como um padrão de representação de informações, através da apresentação da estrutura de um documento XML e de seus componentes.

Dois conceitos essenciais à linguagem XML são: *documento estruturado* e *linguagem de marcação.*

2.1.1 Documento Estruturado

No "mundo digital", um documento é formado por três componentes distintos [MAC 99]:

- **Conteúdo de Dados**: são as palavras e outros elementos (figuras, tabelas, caixas de texto, entre outros) que formam o conteúdo semântico do documento;
- **Estrutura**: trata da organização dos elementos no documento, incluindo regras de integridade para os elementos e a ordem em que eles devem se apresentar;
- **Apresentação**: é a forma pela qual o conteúdo do documento é apresentado.

O conteúdo e a estrutura são os componentes mais importantes. A estrutura do conteúdo permite que o documento possa ser processado por sistemas computacionais, permitindo que o seu conteúdo possa ser interpretado.

2.1.2 Linguagem de Marcação

Uma linguagem de marcação consiste de elementos chamados *marcadores* (*tag*), que têm como finalidade descrever um documento. A descrição pode ser relativa à apresentação,

ou à organização ou ainda ao significado do conteúdo do documento. A linguagem de marcação HTML ("**H**yper **T**ext **M**arkup **L**anguage") é o exemplo mais comum deste tipo de linguagem. Ela é utilizada para a apresentação de conteúdos de documentos na *Web*. Seus marcadores têm a função de definir a aparência das informações quando exibidas em um *browser*[2] (cor, formatação, alinhamento de parágrafo, entre outros), sem existir componentes que representem a semântica destas informações.

2.1.3 O Que é a Linguagem XML?

Segundo [MAC 99], XML é uma linguagem de computador estruturada com a finalidade de descrever informações oferecendo uma estrutura e componentes capazes de identificá-las segundo sua semântica.

XML é um padrão derivado do SGML ("**S**tandard **G**eneralized **M**arkup **L**anguage") - ISO 8879 - criado em 1986 e utilizado nos ramos de negócio como publicações técnicas, indústrias farmacêuticas, aeroespaciais com o objetivo de representar informações através de documentos estruturados [MAC 99]. Por ter sua implementação bastante complexa, a utilização do SGML tornou-se restrita a grande empresas, pois exigia mão-de-obra especializada, o que aumentava o custo. Além disso, documentos SGML eram grandes e complicados, o que dificultava ainda mais a construção de aplicações que suportassem este formato.

XML surge como uma linguagem que tem as características do SGML, porém seus documentos se apresentam de uma forma mais simples, ordenada, de fácil uso e implementação, ao contrário dos documentos SGML, que são geralmente complexos e volumosos [BOS 97].

XML é uma linguagem de marcação, com uma grande diferença em relação à HTML: enquanto que os marcadores da linguagem HTML são fixos, ou seja, são pré-definidos, a XML não define previamente nenhum conjunto de marcadores — eles podem ser criados dinamicamente, de acordo com a necessidade de cada aplicação.

[2] *Software* responsável por tratar documentos HTML e exibi-los para o usuário

HTML

```
<table>
 <h1> Lista de Clientes </h1>
 <tr>
  <td>Nome</td>
  <td>Data de Nascimento</td>
  <td>SEXO</td>
  <td>CPF</td>
 </tr>
 <tr>
  <td>João Silva</td>
  <td>05/10/1977</td>
  <td>Masculino</td>
  <td>12345678911</td>
 </tr>
 <tr>
  <td>José Maria</td>
  <td>09/08/1970</td>
  <td>Masculino</td>
  <td>12369875187</td>
 </tr>
</table>
```

XML

```
<ListaClientes>

  <Cliente>
    <Nome> João Silva </Nome>
    <DataNasc>05/10/1977</DataNasc>
    <Sexo>Masculino</Sexo>
    <CPF>12345678911</CPF>
  </Cliente>

  <Cliente>
    <Nome> José Maria </Nome>
    <DataNasc>09/08/1970</DataNasc>
    <Sexo>Masculino</Sexo>
    <CPF>12369875187</CPF>
  </Cliente>

</ListaClientes>
```

Figura 2.1: Aparência de um documento HTML e de um XML

Nos exemplos de documentos XML e HTML mostrados na figura 2.1, pode-se notar que através dos marcadores `<Nome>`, `<CPF>`, `<DataNasc>`, criados dinamicamente para o documento XML, pode-se identificar as informações contidas no documento. No documento HTML, entretanto, os marcadores são responsáveis apenas por formatar para exibição o seu conteúdo. Caso fosse necessário obter o nome dos clientes envolvidos, o documento HTML não seria capaz de oferecer esta informação, uma vez que o seu conteúdo não está estruturado com este nível de semântica. Em resumo, e de maneira bem diferente de HTML, com XML a definição dos marcadores e como eles estão organizados é uma decisão de quem está criando o documento.

Como um documento XML é estruturado de acordo com as necessidades da aplicação, faz-se também necessário que a estrutura possa ser validada a fim de garantir a confiabilidade do documento. A validação consiste da verificação dos nomes dos marcadores, da opcionalidade de cada marcador no documento, do número de vezes que cada marcador aparece, e da própria estrutura do marcador. A validação pode ser feita por meio de um

apêndice ao documento XML, chamado de DTD ("**D**ocument **T**ype **D**efinitions"). Um DTD consiste de uma sublinguagem que define as regras para a organização dos marcadores de um documento XML. Um DTD pode estar embutido em um documento XML ou definido em um documento à parte. Esta última opção é uma maneira mais usual de utilização de DTDs (apêndice propriamente dito). A utilização de um DTD em um documento XML não é obrigatória, porém a sua não utilização pode gerar documentos XML com uma estrutura diferente da esperada pela aplicação, comprometendo o tratamento das informações.

```
<!DOCTYPE clientes SYSTEM "clientes.dtd"
<ListaClientes>
  <Cliente>
    <Nome> João Silva </Nome>
    <DataNasc>05/10/1977</DataNasc>
    <Sexo>Masculino</Sexo>
    <CPF>12345678911</CPF>
  </Cliente>
  <Cliente>
    <Nome> José Maria </Nome>
    <DataNasc>09/08/1970</DataNasc>
    <Sexo>Masculino</Sexo>
   <CPF>12369875187</CPF>
 </Cliente>
</ListaClientes>
```

Figura 2.2: Documento XML que utiliza um DTD

A figura 2.2 mostra um documento XML que utiliza um DTD definido em um documento externo representado pelo arquivo *clientes.dtd*. Na figura 2.3 pode ser observado o conteúdo deste DTD. Nele estão definidos por exemplo, os elementos que podem aprecer dentro do documento da figura 2.2, como também a ordem e a quantidade de ocorrências destes elementos.

```
<!ELEMENT ListaClientes (Cliente+)>
<!ELEMENT Cliente (Nome, DataNasc, Sexo, CPF)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT DataNasc (#PCDATA)>
<!ELEMENT Sexo (#PCDATA)>
<!ELEMENT CPF (#PCDATA)>
```

Figura 2.3: Definição de um DTD

2.2 Elementos que formam um documento XML

Um documento XML é formado por um conjunto de elementos, que são utilizados para representar informações. Os principais elementos são: marcadores de início e fim, atributos e modelo de conteúdo.

2.2.1 Marcadores de Início e Fim

Os marcadores de início e fim são marcações criadas conforme a necessidade da aplicação. Uma marcação é conhecida como um **elemento** ou **nó** na nomenclatura XML. O conjunto de elementos formam um documento estruturado XML e através deles as informações podem ser identificadas.

```
<ListaClientes>
  <Cliente>
    <Nome> João Silva </Nome>
    <DataNasc>05/10/1977</DataNasc>
    <Sexo>Masculino</Sexo>
    <CPF>12345678911</CPF>
  </Cliente>
</ListaClientes>
```

Figura 2.4: Marcadores de ínicio e fim de um documento XML

Através da marcação `<Nome>`, do exemplo da figura 2.4, é possível extrair o nome de um determinado cliente que aparece na lista de clientes de uma empresa. Os elementos são estruturados de forma a se ter uma hierarquia entre eles o que facilita o entendimento e a manipulação das informações do documento, como pode ser observado na figura 2.5.

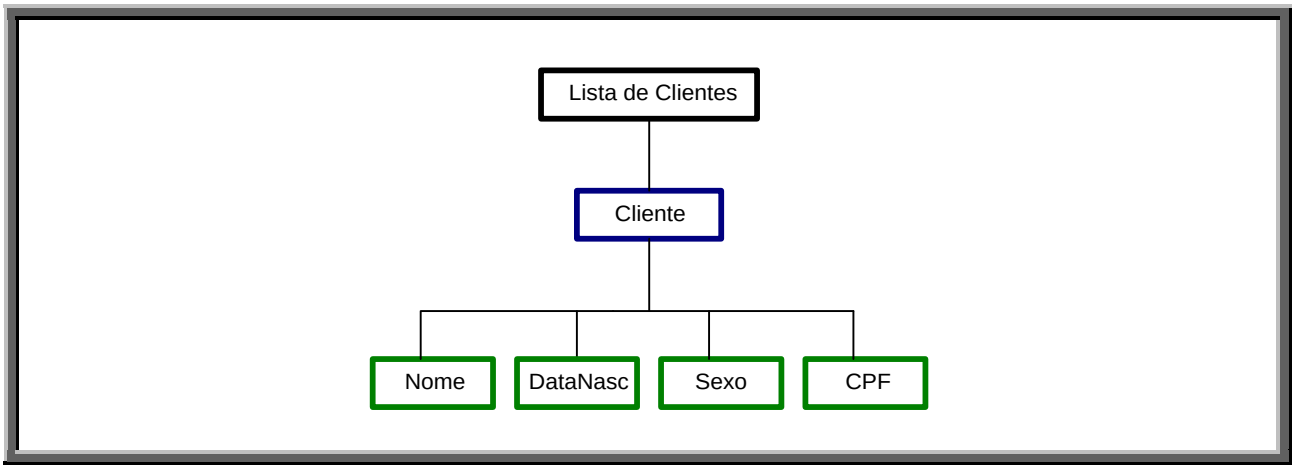

Figura 2.5: Hierarquia de um documento XML

2.2.2 Atributos

Atributos são informações que são adicionadas aos elementos, de forma a especificar valores que correspondam a propriedades particulares para cada um deles.

```
<ListaClientes>
  <Cliente codigo=”0001”>
    <Nome> João Silva </Nome>
    <DataNasc>05/10/1977</DataNasc>
    <Sexo>Masculino</Sexo>
    <CPF>12345678911</CPF>
  </Cliente>
</ListaClientes>
```

Figura 2.6: Atributos dos marcadores de um documento XML

Cada marcação `<Cliente>`, do exemplo da figura 2.6, contém um atributo “codigo” que indica o código de cada cliente da lista de clientes. Este valor é definido através do sinal de atribuição “=”.

Os atributos geralmente são utilizados para descrever informações pequenas [MAC 99]. A utilização de atributos para representar informações extensas não é adequada. Tal manipulação pode comprometer a legibilidade do documento XML.

2.2.3 Modelo de Conteúdo

São informações incluídas entre as marcações de início e fim, que representam o conteúdo de um elemento. O conteúdo de um elemento pode ser: **conteúdo com apenas elementos**, **conteúdo misto**, **conteúdo com apenas texto**, **conteúdo vazio** ou **qualquer conteúdo**. O tipo do conteúdo de um elemento é definido nas declarações do DTD associado ao documento.

Conteúdo com apenas elementos

Este modelo de conteúdo é usado quando elementos podem conter apenas outros elementos, como mostra o exemplo da figura 2.7.

DTD
<!ELEMENT Cliente (Nome, Telefone)>

XML
<Cliente>
  <Nome>João Silva</Nome>
  <Telefone> 221-2589 </Telefone
</Cliente>

Figura 2.7: Modelo de conteúdo com apenas elementos

Conteúdo misto

Este tipo de modelo permite que o elemento possua em seu conteúdo qualquer elemento, em qualquer ordem ou quantidade, permitindo ainda que textos sejam incluídos. O exemplo da figura 2.8 exibe um documento XML onde o elemento `Clientes` é formado por um modelo de conteúdo misto.

```
DTD
<!ELEMENT Cliente(#PCDATA | Nome | Telefone )* >

XML
<Cliente>
  Estas são as informações do cliente
  <Nome>João Silva</Nome>
  <Telefone> 221-2589 </Telefone>
  <Telefone> 231-4050 </Telefone>
</Cliente>
```

Figura 2.8: Modelo de conteúdo misto

Conteúdo com apenas texto

No modelo de conteúdo com apenas um texto, um elemento pode possuir apenas texto, não sendo permitido outros elementos. A figura 2.9 exemplifica a utilização deste tipo de modelo de conteúdo.

```
DTD
<!ELEMENT Cliente(#PCDATA)>

XML
<Cliente> João Silva </Cliente>
```

Figura 2.9: Modelo de conteúdo com apenas texto

Conteúdo vazio

Este tipo de modelo de conteúdo não permite que o elemento possua algum tipo de conteúdo, seja um outro elemento ou um texto. Este conteúdo é representado sem as marcações de início e fim, mas com apenas uma marcação como mostra a figura 2.10.

```
DTD
<!ELEMENT Cliente EMPTY>

XML
<Cliente/>
```

Figura 2.10: Modelo de conteúdo vazio

Qualquer conteúdo

Este tipo de modelo de conteúdo permite que qualquer elemento ou texto seja incluído dentro do conteúdo de um elemento, sem definir regras de ordem ou quantidade de ocorrências. A figura 2.11 mostra a utlização deste tipo de modelo de conteúdo.

DTD
<!ELEMENT Cliente ANY >

XML
<Cliente>
Estas são as informações do cliente
<Nome>João Silva</Nome>
Telefone residencial
<Telefone> 221-2589 </Telefone>
Telefone comercial
<Telefone> 231-4050 </Telefone>
</Cliente>

ou

<Cliente>João Silva</Cliente>

ou

<Cliente>
<Nome>João Silva</Nome>
</Cliente>

Figura 2.11: Modelo de conteúdo com qualquer elemento

Este modelo é semalhante ao modelo de conteúdo misto, com a diferença que ele não defini quais elementos podem aparecer no conteúdo.

2.3 Representando Informações com XML

Antes de utilizar XML para representar informações, é necessário entender quais tipos de informações serão modeladas.

As informações modeladas em XML, estão divididas em dois grandes grupos:

- textos de documentos;

- dados.

Segundo [WIL 00] a diferença entre textos e dados consiste no fato de que dados são informações menores e a alteração na ordem de seus elementos não compromete o seu significado final.

2.3.1 XML para Descrição de Texto

A representação de partes de livros e artigos para um formato eletrônico é uma das principias utilizações da linguagem XML.

```
<paragrafo capitulo="XML – Um sumário executivo" pagina="6">
    XML é uma linguagem de computador com a finalidade de descrever informações.
Quer oferece simplicidade e transportabilidade para as aplicações que utilizam esta
linguagem.
<paragrafo>
```

Figura 2.12:XML para descrever texto

No trecho de documento XML do exemplo da figura 2.12, é descrito um parágrafo de um capítulo de um livro sobre XML. A informação representada por este documento é classificada como um texto, pois se enquadra nas seguintes características:

- são informações extensas;
- possuem conteúdo explicativos e descritivos;
- a alteração na ordem com que as palavras aparecem compromete o significado final da informação[WIL 00];
- a informação não perde o significado quando a marcação é retirada[WIL 00].
- não permitem uma análise mais profunda da informação recuperada.

2.3.2 XML para Descrever Dados

No exemplo a seguir, um documento XML foi estruturado com o objeto de realizar a representação dos dados de uma determinada venda.

```
<Venda>
 <NumeroVenda>1425698</NumeroVenda>
 <DataVenda>05/01/2000</DataVenda>
 <ValorVenda>500,00</ValorVenda>
 <ItemVenda>
   <DescricaoItem>Placa de Fax Modem 57.600 bps </DescricaoItem>
   <ValorItem>100,00</ValorItem>
   <QuantidadeItem>5</QuantidadeItem>
 </ItemVenda>
</Venda>
```

Figura 2.13: XML para descrever dados

No documento XML da figura 2.13, o número da venda, representado pela marcação, `<NumeroVenda>`, pode aparecer depois do valor da venda (representado pela marcação `<ValorVenda>`), sem prejudicar o entendimento do contexto.

As principais vantagens de representar informações em documentos estruturados através de dados, são:

- sistemas computacionais podem realizar tarefas que necessitam extrair e analisar informações específicas do contexto;
- representar dados oriundos de um banco de dados.

Utilizando o exemplo anterior, um sistema computacional seria capaz de obter a data em que a venda de número 1425698 foi realizada, operação que não seria possível se a venda fosse representada através de um texto como no exemplo da figura 2.14.

<Venda>
A venda de número, 1425698, foi realizada em 05/01/2000 no valor de 500 reais e foram vendidos cinco unidades de Placas Fax Modem 57.600bps no valor de 100 reais cada.
</Venda>

Figura 2.14: Documento XML representando um venda de forma textual

2.3.3. Representando Dados em XML

A representação de dados através de XML pode ser feita através da utilização direta de elementos ou através dos atributos dos elementos.

Representando Dados com Elemento

A utilização de elementos para representar dados em documentos XML consiste no uso de dois tipos de elementos presentes na sintaxe da linguagem XML:

- elemento que contém outros elementos*;*
- elemento que contém apenas texto.

Elemento que Contêm outros Elementos

Este tipo aparece quando um elemento possui em seu conteúdo outros elementos. Utilizado quando uma informação (conjunto de dados), para ser bem representada, necessita de uma estruturação que abrange outras informações.

```
<Venda>
 <DataVenda>24/03/2001</DataVenda>
 <ValorVenda>30</ValorVenda>
 <Cliente>
   <CodigoCliente>12598</CodigoCliente>
   <NomeCliente>Jõao Silva</NomeCliente>
 </Cliente>
 <ItemVenda>
   <NumeroItem>1425</NumeroItem>
   <ValorItem>30,00</ValorItem>
   <QuantidadeItem>1</QuantidadeItem>
 </ItemVenda>
</Venda>
```

Figura 2.15: XML com elemento que contém outros elementos

No XML da figura 2.15, para que a informação de venda possa ser bem representada, é necessário que as informações relativas ao cliente e aos itens vendidos sejam modeladas como uma informação que compõe a venda.

Elemento que Contêm apenas Texto

São elementos que possuem em seu conteúdo somente dados, como o exemplo da figura 2.16.

```
<NomeCliente> João </NomeCliente>
<Idade> 31 </Idade>
```

Figura 2.16: XML com elementos que contém apenas texto

São elementos capazes de representar os dados em seu estado mais primitivo. São bastante utilizados para representar valores de colunas de uma tabela de um banco de dados relacional.

Representando dados com Atributos das Marcações

Da mesma forma que os elementos com apenas texto podem representar dados em documentos XML, os atributos também podem ser utilizados para este fim. Quando se deseja representar dados, o projetista do XML vai ter que escolher entre elementos ou atributos, uma vez que, o uso das duas formas em um mesmo documento não se apresenta como uma boa solução, pois compromete a legibilidade[WIL 00]. O exemplo da figura 2.17 mostra como os dados podem ser modelados utilizando elementos e atributos.

```
Com Elemento
<Cliente>
  <PrimeiroNome> João </PrimeiroNome>
  <SegundoNome> Carlos </SegundoNome>
</Cliente>

Com atributos
<Cliente PrimeiroNome = “João” SegundoNome = “Carlos” />
```

Figura 2.17: Representando dados através de elementos e através de atributos

2.4. Interpretadores ou Processadores XML

Interpretadores ou Processadores XML (*XML Parser*), são *softwares* que tratam documentos XML, permitindo às aplicações, que utilizem este tipo de documento, possam trabalhar com as informações neles contidas. É responsável por identificar as marcações e os atributos, obtendo os dados e disponibilizando-os para aplicações que irão utilizá-los[MAC 99]. Caso exista um DTD vinculado ao documento, o processador XML deve fazer a validação do XML conforme as regras definidas no DTD. Além disso, um processador XML

deve permitir que as aplicações possam adicionar, atualizar e excluir dados no documento XML.

Os processadores devem implementar a manipulação de documentos XML conforme a especificação de duas APIs ("**A**pplication **P**rogramming **I**nterface"), DOM[3] ("**D**ocument **O**bject **M**odel") e SAX ("**S**imple **A**PI for **X**ML"), que definem como deve ser realizado o processamento e como os dados serão manipulados pelas aplicações. A definição destas APIs tem a finalidade de padronizar a interação com documentos XML, funcionando como uma camada intermediária entre o processador e o XML [WIL 00]. A implementação destas APIs pode ser feita utilizando qualquer linguagem de programação (*Visual Basic*, *Java*, *C++*, *Perl*), desde de que esteja de acordo com a especificação da API utilizada.

A API DOM foi especificada pelo W3C ("**W**ord **W**ide **W**eb **C**onsortium")[4], enquanto que SAX foi especificada por membros do *XML-Dev*[5]. A grande diferença entre estas APIs consiste na estratégia de processamento dos documentos XML. Quando um *XML Parser*, baseado na API DOM, processa um documento XML, ele armazena em memória uma árvore que representa as marcações presentes no documento organizadas hierarquicamente, como pode ser visto na figura 2.5. Com isso a aplicação pode "caminhar" por esta árvore obtendo as informações necessárias. Já a API SAX trabalha através de eventos. Quando é encontrada uma marcação de início o SAX avisa à aplicação, o mesmo acontece quando a marcação de fim é encontrada. Entre o evento da marcação de início e da marcação de fim pode-se recuperar a informação contida entre elas. A vantagem da utilização da API DOM consiste no fato de que esta permite a adição, atualização e exclusão das informações do documento, como também a aplicação de filtros de consulta[WIL 00].

2.5 Linguagens de Consulta para documentos XML

Aplicações que utilizam documentos XML para representar informações, muitas vezes necessitam recuperá-las através de pesquisas mais refinadas, aplicando filtros e fazendo diferentes seleções. O W3C definiu uma especificação para a construção de linguagens de consulta para documentos XML[6]. Esta especificação se preocupa com a definição de um

[3] www.w3c.org/TR/DOM-Level-2
[4] Grupo responsável por definir padrões para Internet: www.w3c.org
[5] Grupo de desenvolvimento em XML que tem como líder David Megginson: www.megginson.com/SAX/index.html
[6] www.w3c.org/XML/Query

modelo teórico sem fazer referência a detalhes de implementação [WIL 00]. Desta forma algumas linguagens de consulta para documentos XML foram desenvolvidas, como por exemplo XQL.

XQL é uma linguagem de consulta para documentos XML que permite recuperar informações através de uma determinada sentença, com o uso de operadores de seleção, lógicos e de comparação. O resultado de uma consulta em XQL, pode ser um outro documento XML que deverá conter um ou mais elementos resultantes da consulta[XQL 98]. Por exemplo, a sentença XQL "**/Venda[/Venda/Cliente/CodigoCliente='12598']**" atuando sobre o documento XML da figura 2.12, vai retornar um outro documento XML contendo apenas os elementos que representam as vendas efetuadas pelo cliente cujo código é igual a 12598. Para realizar o processamento de uma consulta XQL, deve-se fazer uso de uma **máquina de consulta XQL** ou simplesmente *XQL Engine*, capaz de interpretar uma senteça XQL e disponibilizar um resultado.

Segundo [WIL 00], XQL está para XML assim como SQL está para os banco de dados relacionais.

2.6 XML para o Compartilhamento de Informações

A arquitetura de um sistema de compartilhamento de dados consiste basicamente na troca de mensagens estruturadas entre aplicações ou entre banco de dados, a fim de enviar e receber dados. A estruturação destas mensagens consiste no principal desafio que deve ser superado para a construção deste tipo de sistema [SEL 01]. XML aparece como a solução adequada para estruturar estas mensagens devido as seguintes características [QUE 01, XPE 00]:

- **Simplicidade:** sua sintaxe é simples, facilitando o trabalho de definição das informações contidas no documento.
- **Facilidade de uso e independência de plataforma:** documentos XML são simples arquivos textos que podem ser lidos por qualquer ser-humano em qualquer editor de texto, e processados por *softwares* (*XML Parse*) que utilizam APIs padrões de processamento dos documentos, o que possibilita sua utilização em qualquer plataforma[SUN2 01, UEY 00]. .

- **Padrão para representação de informações:** através dos marcadores é possível identificar e dar significado as informações. Desta forma, as informações podem ser identificadas e analisadas por sistemas computacionais. A padronização destas informações será explicada a seguir.
- **Consultas mais dinâmicas:** com a representação das informações através de marcadores, torna-se possível realizar consultas com alto nível de customização utilizando, por exemplo, XQL [UEY 00].

Muitas organizações, mesmo aquelas que competem de forma acirrada no mundo comercial, podem se beneficiar da capacidade de intercâmbio de informações [MAC 99].

A fim de permitir estas operações é necessário que seja adotado um formato padrão entre as organizações de forma a permitir que suas aplicações possam gerar e obter informações uma das outras. A forma clássica de intercâmbio de informações consiste no seguinte contexto: uma organização A se comunica com B segundo um formato definido pelas duas, caso A deseje comunicar-se com C outro formato deverá ser definido, como ilustra a figura 2.18.

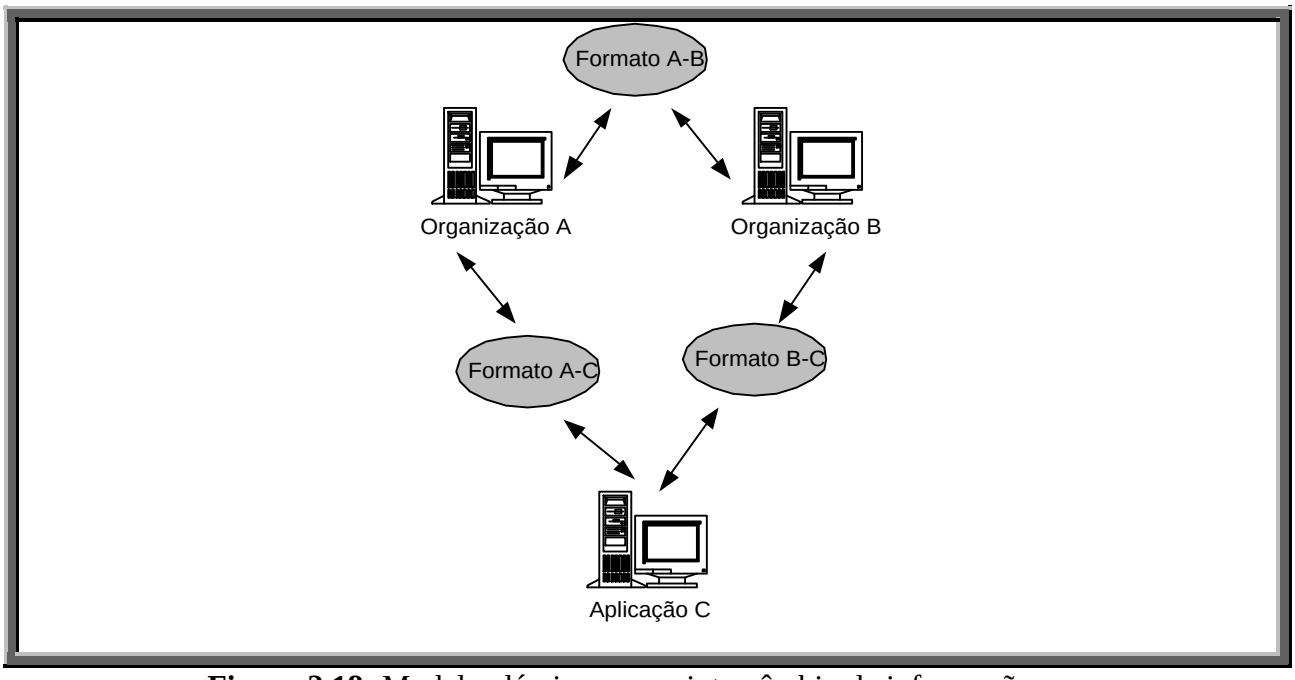

Figura 2.18: Modelo clássico para o intercâmbio de informações

Esta solução implicaria na construção de aplicações proprietárias para cada relacionamento comercial da organização A e na concepção de novas aplicações toda vez que novas parcerias surgissem[MOR 00, MAC 99].

Uma outra solução pode ser adotada pelas organizações envolvidas, a fim de se obter um padrão único para a comunicação entre as organizações. Este solução é chamada de **notação de ponto central**[MAC 99] e pode ser observada na figura 2.19.

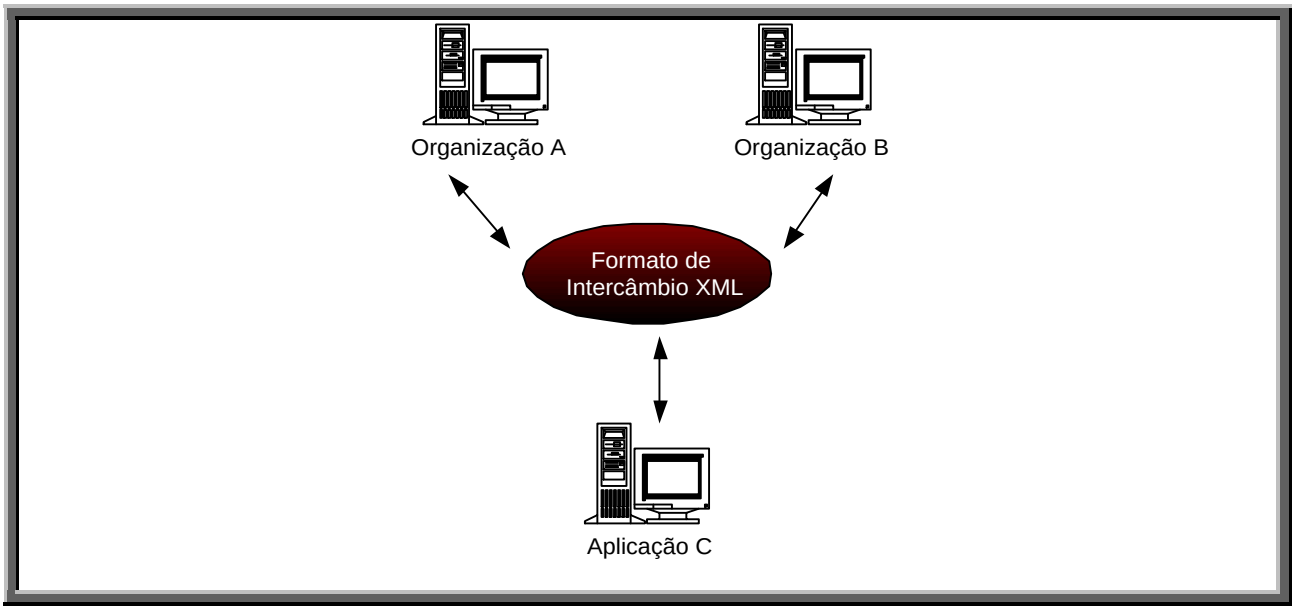

Figura 2.19: Notação de ponto central para o intercâmbio de informações

A utilização da linguagem XML como o ponto central para o intercâmbio de informações vem se tornando a principal solução para o desenvolvimento de aplicações que necessitem deste tipo de operação, devido as caratcterísticas citadas na seção 2.6 [MAC 99].

Alguns padrões veem surgindo, na tentativa de padronizar e facilitar o intercâmbio de informações entre as organizações, entre eles:

- **RosettaNet**[7]: é um consórcio de empresas ligadas a tecnologia da informação, voltadas para a padronização das informações relevantes a esta área. Este arquitetura propõe um dicionário mestre com as definições mais comuns das empresas, produtos e suas transações**.**
- **eCo Framework Project**[8]: proposto pela CoomerceNet – grupo formado por 35 empresas entre elas: *3COM*, *IBM*, *American Express*, *HP*, *NEC*, *Microsoft*, *Intel* entre outras. A grande diferença entre este padrão para o *RosettaNet* é a diversificação da atuação, uma vez que o *RosettaNet* atua apenas em empresas

[7] www.rosettanet.org
[8] www.comerce.net/projects/currentprojects/eco/

ligadas a área da tecnologia da informação e o *eCo Framework Project* atua sobre diversas áreas.

- **ebXML("Enabling Electronic Business with XML")[9]:** tem como objetivo criar um conjunto de padrões e uma arquitetura para operações de *e-businnes* sobre XML, ou seja, permitir que pequenas, médias e grandes empresas possam trocar informações de negócios através da Internet.

2.7 XML e Java

Segundo [SUN201, MOR00], utilizar XML e Java na construção de aplicações de intercâmbio de dados garante a transportabilidade necessária para este tipo de aplicação. Enquanto que XML garante a transportabilidade para as informações, Java garante a transportabilidade para o código.

A integração Java/XML é possível devido ao poderoso suporte oferecido pela plataforma Java à tecnologia XML. Este suporte é obtido através da utilização da API[10] Java para o tratamento de XML, que auxilia o programador no tratamento das informações, fazendo com que este não interajam diretamente com o XML, o que aumentaria a possibilidade de erros de estruturação e de recuperação das informações.

2.7.1 API Java para XML

A plataforma Java disponibiliza duas APIs para o tratamento de informações em documentos XML: JAXP("**J**ava **A**PI for **X**ML **P**rocessing") e JAXB("**J**ava **A**rchitecture for **X**ML **B**inding").

JAXP

JAXP facilita o processamento de informações inseridas em um documento XML para aplicações escritas em Java, através dos métodos disponibilizados por esta API[ARM 01]. Ela permite o processamento do XML através dos padrões SAX e DOM, e foi projetada para

[9] www.ebxml.org
[10] java.sun.com/xml

ser flexível, permitindo o acoplamento de qualquer *parse XML*. Esta flexibilidade chamada de "camada de plugabilidade" permite plugar qualquer implementação das APIs SAX e DOM (ver figura 2.20) . A versão 1.1 da JAXP traz a implementação do *Crimson*, *parser* desenvolvido numa parceria *Sun*[11] e *Apache Software Fundation*[12].

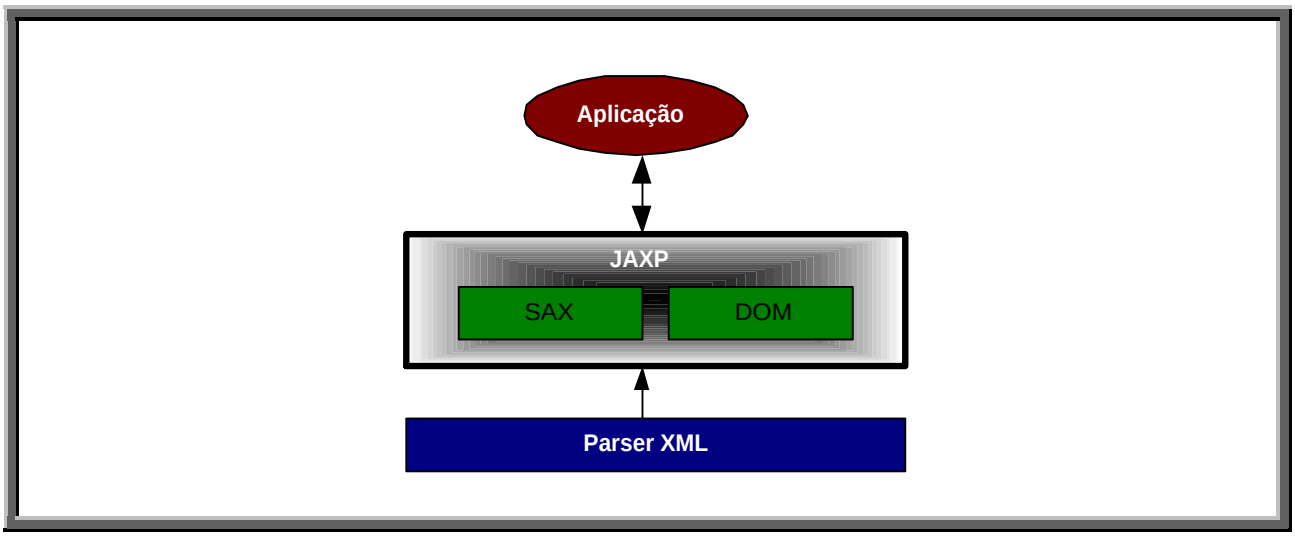

Figura 2.20: Esquema de utilização da API JAXP

JAXB

JAXB faz o mapeamento entre documentos XML e classes Java. Com um determinado DTD o JAXB é capaz de gerar um conjunto de classes Java que representa toda a estrutura de um documento XML[SUN2 01]. O desenvolvedor utiliza os métodos disponibilizados por estas classes para manipular as informações do documento XML (recuperar, adicionar, remover e atualizar).

Fazendo um comparação entre as APIs JAXP e JAXB pode-se observar que a primeira oferece ao programador maior flexibilidade na manipulação de documentos XML, enquanto que a segunda facilita o desenvolvimento de aplicação para o tratamento de documentos XML porém sem grande flexibilidade. Escolher uma destas API é uma decisão do programador que deve verificar o nível de flexibilidade necessária para sua aplicação.

[11] Empresa que criou a linguagem Java
[12] Organização que desenvolve soluções para Internet: www.apache.org

2.8 Conclusão

Diante do exposto neste capítulo, pode-se verificar que a linguagem XML aparece como solução para a padronização da estrutura da informação contida em documentos, a fim de permitir o intercâmbio de informações. Isto se deve ao fato de XML ser uma linguagem de marcação, onde os marcadores podem ser criados conforme a necessidade representativa da informação, simples e independente de plataforma. Estas características permitem que a linguagem XML possa ser empregada na construção de aplicações que realizem o intercâmbio de dados, como em transações *businnes-to-businnes* ou na troca de dados entre diferentes bancos de dados.

A combinação da linguagem Java com XML, através das APIs JAXP e JAXB, possibilita a criação de aplicações capazes de compartilhar informações com um alto nível de transportabilidade. Enquanto Java oferece transportabilidade para o código, XML oferece transportabilidade para as informações envolvidas.

Capítulo 3

Trabalhos Relacionados: Modelos de Mapeamento de Banco de Dados através de XML e Sistemas de Intercâmbio de Dados entre Banco de Dados utilizando XML

3.1 Introdução

Sistemas de intercâmbio de dados entre banco de dados (SIBD), são aplicações que realizam a troca de dados entre banco de dados, geralmente criados e mantidos por Sistemas de Gerência de Banco de Dados (SGBD), como *Oracle* da *Oracle Corporation*[13], *SQL Server* da *Microsoft Corporation*[14], *Informix* e *DB2* da IBM[15], *InterBase* da *Borland*[16], entre outros.

Os SIBDs geralmente são utilizados para os seguintes tipos de aplicação:

- **Duplicação de dados:** banco de dados diferentes que necessitam estar sincronizados em relação aos dados.
- **Disponibilização de dados de sistemas legados:** permitir que outras aplicações possam acessar dados de sistemas legados [TUR 00].
- **Integração de aplicações:** possibilitar que aplicações que utilizem diferentes tecnologias de banco de dados ou de organizações diferentes possam compartilhar seus dados [MOR 00].

Em relação à arquitetura básica de um SIBD, podemos separá-la em dois componentes principais:

- **Documento de dados:** documento que contém os dados estruturados provenientes de um banco de dados.

[13] www.oracle.com
[14] http://www.microsoft.com/sql/
[15] http://www-3.ibm.com/software/data/db2/
[16] http://www.borland.com/interbase/

- **Interpretador do documento:** é o núcleo do SIBD, ele extrai os dados de um banco de dados estruturando-os em um documento e obtém estes dados incluindo-os em uma outra base de dados.

Com base nesta arquitetura, o funcionamento de um SIBD consiste basicamente na ação de exportação dos dados de uma base de dados (banco de dados chamado de origem) para um documento padrão e da importação dos dados do documento padrão para uma outra base de dados (banco de dados chamado de destino). A figura 3.1 define graficamente este esquema.

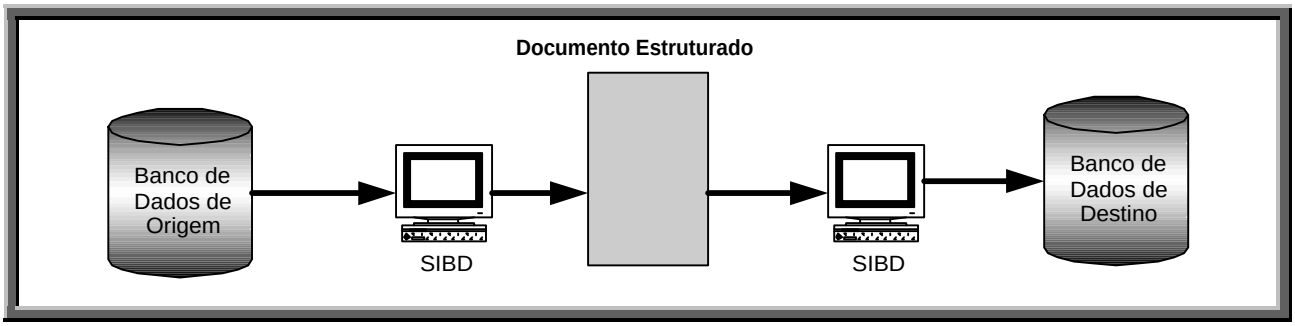

Figura 3.1: Esquema básico de funcionamento de um SIBD

3.2 Requisitos de um SIBD

Para que um SIBD possa ser utilizado com sucesso nas aplicações ele deve ser construído segundo os seguintes requisitos funcionais:

1) Estruturar os Dados em um Documento Padrão

Um SIBD deve utilizar-se de um padrão de estruturação de dados provenientes do banco de dados para que estes possam ser posteriormente interpretados e utilizados de forma correta. A operação de estruturação dos dados para o documento consiste no mapeamento destes dados e do esquema no qual estão inseridos, incluindo-se os meta-dados, para um determinado formato segundo uma linguagem de intercâmbio que ofereça ao mesmo tempo legibilidade e riqueza na representação de informações.

2) Suporte a Diferentes SGBDs

Para que um SIBD seja utilizado com sucesso, é necessário que ele ofereça suporte a diferentes SGBDs. Por exemplo, uma empresa A, que utiliza o SGBD *Oracle*, pode em um determinado momento necessitar intercambiar dados com uma empresa B, que utiliza o SGBD *SQL Server*. O SIBD que for aplicado nesta operação deve oferecer suporte tanto para o SGBD da empresa A como da empresa B.

3) Suporte a Diferentes Plataformas

Ao mesmo tempo que um SIBD deve oferecer acesso a diferentes SGBDs, ele também deve atuar em diferentes sistemas operacionais. Utilizando o mesmo exemplo usado para justificar o requisito número 1, a empresa A pode utilizar um sistema operacional diferente do utilizado na empresa B, como *UNIX* e *Windows* da *Microsoft* respectivamente. Assim o SIBD deve poder ser utilizado tanto para plataforma da empresa B, como da empresa A.

4) Seleção de Dados para Importação/Exportação

O SIBD deve permitir que os dados que serão exportados de um banco de dados possam ser selecionados (por exemplo, através de sentenças SQL) oferecendo maior flexibilidade nas operações de intercâmbio de dados. A mesma necessidade de seleção se apresenta no momento da importação dos dados

5) Disponibilizar uma API para o Desenvolvimento de Aplicações

Faz-se necessário que o SIBD disponibilize uma API que permita ao desenvolvedor incorporar em suas aplicações a capacidade de realizar intercâmbio de dados.

Além dos requisitos funcionais apresentados, alguns requisitos não-funcionais podem ser incorporados ao SIBD, por exemplo: disponibilizar uma *interface* gráfica para facilitar o uso do SIBD e oferecer funções capazes de tratar os dados para apresentação, como por exemplo: transformar o documento que contém os dados em documentos HTML para exibição na *Web*.

3.3 Desafios para a Construção de um SIBD

Na construção de um SIBD, o desenvolvedor tem que superar alguns desafios de forma a atender os seus requisitos funcionais.

- **Primeiro desafio**

O primeiro desafio a ser superado é tornar o SIBD independente de plataforma e de SGBD, atendendo ao primeiro e ao segundo requisito. Adotar tecnologias que proporcionem esta transportabilidade, no momento do seu desenvolvimento, é a solução para superar este desafio.

- **Segundo desafio**

O segundo desafio consiste em como mapear informações de um determinado banco de dados para um documento estruturado [TUR00, SHA00, QUE01]. O documento deve ser capaz de representar todo o esquema do banco de dados e os dados propriamente ditos. Por exemplo, a tabela A de um banco de dados relacional deve ter sua estrutura mapeada, colunas e tipos das colunas e os relacionamentos com outras tabelas, e as linhas que preenchem esta tabela.

Através destas informações, o SIBD deve ser capaz de identificar os dados e relacioná-los com a estrutura do banco de dados de destino.

- **Terceiro desafio**

A heterogenidade de estrutura e de semântica dos dados é o terceiro desafio a ser superado [SEL 01, LIV 00].

Os banco de dados de origem e destino podem ter seus dados estruturados de maneiras diferentes. Por exemplo, uma determinada informação no banco de dados de origem poderia ser representada por uma única tabela relacional, enquanto que no banco de dados de destino esta mesma informação poderia ser representa por duas tabelas.

O mesmo pode acontecer com a semântica dos dados. Por exemplo, no banco de dados de origem a informação de velocidade de um determinado carro poderia ser medida em

km/h, enquanto que no banco de dados de destino esta mesma informação poderia ser medida em milhas/h. Para resolver esses conflitos estruturais e/ou semânticos, o SIBD deve adotar mecanismos que ofereçam flexibilidade para que o usuário determine como resolvê-los.

3.4 Mapeamento de Banco de Dados Utilizando XML

Transportabilidade, flexibilidade e padronização para representar informações são características que colocam a linguagem XML como a melhor solução para o mapeamento do esquema de banco de dados para um documento estruturado [TUR 00] A transportabilidade auxilia a superação do terceiro desafio e a cumprir o segundo requisito de um SIBD, a flexibilidade possibilita organizar as informações conforme o esquema do banco de dados, enquanto que a padronização para representar as informações oferece facilidade no momento de identificá-las [WIL 00, WEB 01]

3.4.1 Mapeamento do Modelo Relacional para XML

O modelo relacional já clássico consiste basicamente na representação de um banco de dados como um conjunto de tabelas relacionadas ou não, onde cada tabela é descrita por um conjunto fixo de colunas e cada linha ou registro da tabela é constituída por valores destas colunas [WIL 00, TUR 01, BOS 97].

[WIL 00] apresenta duas formas de mapeamento de tabelas relacionais para XML: usando **elementos** e usando **atributos**. Este modelo representa apenas a estrutura da tabela e os dados, excetuando-se a representação do tipo e tamanho das colunas. A maioria dos SIBDs utilizam-se deste modelo de mapeamento.

Representando Registros e Colunas de Tabelas

Os registros são representados através de **elementos**, e as colunas ou por meio de **elementos com apenas texto** (estruturados dentro do **elemento** que representa o registro), ou através de **atributos** do **elemento** representativo do registro.

A definição de uma tabela de clientes da figura 3.2 será utilizada para exemplificar um mapeamento.

Tabela de Clientes		
Coluna	**Tipo**	**Tamanho**
Código	Número	10
PrimeiroNome	Alfanumérico	30
SegundoNome	Alfanumérico	30
Endereço	Alfanumérico	50
Telefone	Alfanumérico	20

Figura 3.2: Definição da tabela-exemplo Clientes

Mapeando Colunas Utilizando Elementos

A figura 3.3 mostra o DTD e o XML que representam as colunas de um modelo relacional utilizando elementos.

Definição do DTD

```
<!ELEMENT Cliente ( Codigo, PrimeiroNome, SegundoNome, Endereco,
                    Telefone) >
<!ELEMENT Codigo (#PCDATA)>
<!ELEMENT PrimeiroNome (#PCDATA)>
<!ELEMENT SegundoNome (#PCDATA)>
<!ELEMENT Endereco (#PCDATA)>
<!ELEMENT Telefone (#PCDATA)>
```

Definição do XML

```
<Cliente>
  <Codigo>354676</Codigo>
  <PrimeiroNome>José</PrimeiroNome>
  <SegundoNome>Silva</SegundoNome>
  <Endereco>Rua A n. 36</Endereco>
  <Telefone>222-2352</Telefone>
</Cliente>
```

Figura 3.3: Mapeamento de colunas de um modelo relacional utilizando elementos

Cada elemento com apenas texto (`Codigo`, `PrimeiroNome`, `SegundoNome`, `Endereco`, `Telefone`) vai representar um determinado campo do registro, enquanto que o elemento `Cliente` será formado por todos os elemento que representam os campos, obtendo-se desta forma um registro da tabela de clientes.

Mapeando Colunas Utilizando Atributos

A figura 3.4 mostra o mapeamento de colunas de um modelo relacional utilizando atributos.

Definição do DTD

```
<!ELEMENT Cliente >
<!ATTLIST Cliente
   Codigo CDATA #REQUERIED
   PrimeiroNome CDATA #REQUERIED
   SegundoNome CDATA #REQUERIED
   Endereco CDATA #REQUERIED
   Telefone CDATA #REQUERIED >
```

Definição do XML

```
<Cliente Codigo=”354676” PrimeiroNome=”José” SegundoNome=”Silva”
         Endereco=”Rua A n. 36” Telefone=”222-2352” />
```

Figura 3.4: Mapeando colunas de um modelo relacional utilizando atributos

Representando o conteúdo de uma tabela

A figura 3.5 mostra a definição XML para representar o conteúdo de uma tabela relacional através de elementos e através de atributos.

Definição do DTD

```
<!ELEMENT Clientes (Cliente*)>

<!ELEMENT Cliente >
<!ATTLIST Cliente
   Codigo CDATA #REQUERIED
   PrimeiroNome CDATA #REQUERIED
   SegundoNome CDATA #REQUERIED
   Endereco CDATA #REQUERIED
   Telefone CDATA #REQUERIED >
```

Definição do XML

Através de Atributos

```
<Clientes>
  <Cliente Codigo=”354676” PrimeiroNome=”José” SegundoNome=”Silva”
           Endereco=”Rua A n. 36” Telefone=”222-2352” />
  <Cliente Codigo=”123589” PrimeiroNome=”Carlos” SegundoNome=”Augusto”
           Endereco=”Rua B S/N” Telefone=”214-0590” />
  <Cliente Codigo=”101020” PrimeiroNome=”Amadeus” SegundoNome=”Bueno”
           Endereco=”Rua Portugal n. 20” Telefone=”999-2519” />
</Clientes>
```

Através de Elementos

```
<Clientes>
  <Cliente>
    <Codigo>354676</Codigo>
    <PrimeiroNome>José</PrimeiroNome>
    <SegundoNome>Silva</SegundoNome>
    <Endereco>Rua A n. 36</Endereco>
    <Telefone>222-2352</Telefone>
  <Cliente>
  <Cliente>
    <Codigo>123589</Codigo>
    <PrimeiroNome>Carlos</PrimeiroNome>
    <SegundoNome>Augusto</SegundoNome>
    <Endereco>Rua B S/N</Endereco>
    <Telefone>214-0590</Telefone>
  <Cliente>
</Clientes>
```

Figura 3.5: Representando o conteúdo de uma tabela relacional através de elementos e através de atributos

A marcação `<Clientes>`, do XML da figura 3.5, representa toda a tabela de clientes, que contém várias ocorrências da marcação `<Cliente>`, que representam os vários registros desta tabela.

Vantagens da representação por atributos

A representação através de atributos se apresenta mais vantajosa em relação à representação por elementos, principalmente no que diz respeito aos seguintes aspectos [WIL 00]:

- compatibilidade com banco de dados;
- tamanho do documento.

Em a relação legibilidade, a representação por atributo não obtém vantagens sobre a representação por elementos, uma vez que em ambos os casos os documentos gerados são claros e permitem que o seu conteúdo possa ser facilmente interpretado.

Compatibilidade com banco de dados

Em um banco de dados relacional a estrutura da base de dados é formada por tabelas e relacionamentos e os dados são representados através de colunas, ou seja, a estrutura e os dados são representados através de elementos distintos. Quando se deseja utilizar documentos XML para representar banco de dados relacionais, o XML deve ter uma aparência que seja um espelho deste modelo. Desta forma, ao se utilizar representação através de atributos, o documento XML fica mais próximo da estrutura do banco de dados, uma vez que as tabelas são representadas por elementos e as colunas por atributos. Já na representação através de elementos, as tabelas e colunas são representadas utilizando o mesmo elemento não havendo uma distinção clara do que é estrutura e do que é dado. Isto pode ocasionar dificuldades no momento de extrair estas informações[WIL 00].

Tamanho do documento

A utilização de elementos para representar tabelas e colunas implica em documentos extensos, uma vez que, para todo valor representado das colunas serão necessários os marcadores de início e fim. Já o uso de atributos para representar os dados das colunas levaria a construção de documentos menores, pois não se utilizariam marcadores de início e fim.

O tamanho do documento e a performance estão relacionados. Quanto maior o tamanho do documento e maior o número de elementos, maior o tempo necessário para processar um documento XML.

3.4.2 Mapeamento do Modelo a Objeto Padrão ODMG para XML

Para realizar o mapeamento de um banco de dados orientado a objeto (OO) para XML, é necessário que todos os elementos presentes no modelo do banco de dados possam ser representados, de forma a permitir a estruturação dos dados em documentos XML. O modelo de dados OO a que nos referimos é o padrão ODMG[17] ("**O**bject **D**ata **M**anagement **G**roup") [CAT 00].

O Padrão ODMG de Banco de Dados OO

O Comitê ODMG — "**O**bject **D**ata **M**anagement **G**roup" — define uma especificação para a implementação de dois tipos de produtos: SGBDs Orientados a Objetos (SGBD OO) e "Object-to-Database Mappings" (ODMs) [CAT 00]. SGBDs OO tratam diretamente de objetos persistentes. ODMs armazenam objetos em SGBDs Relacionais, através de mapeamentos apropriados. Estes dois tipos são classificados como "Object Data Management Systems" (ODMS).

A especificação ODMG consiste, entre outras, de três linguagens, uma para representar os objetos, outra para o intercâmbio de objetos e uma para a consulta e atualizações a objetos, todas especificadas para utilização em ODMSs.

- "**O**bject **D**efinition **L**anguage" (ODL) é utilizada para descrever esquemas de bancos de dados orientados a objeto.

- "**O**bject **I**nterchange **F**ormat" (OIF) é uma linguagem de intercâmbio de objetos, especificada para realizar extração de informações de um SGBD OO para um arquivo "stand-alone" padronizado, e vice-versa.

[17] www.odmg.org

- "**O**bject **Q**uery **L**anguage" (OQL) é uma linguagem declarativa para consulta e manutenção a objetos armazenados em um SGBD OO. O padrão para consulta a dados relacionais (SQL) foi a base para a concepção desta linguagem.

O padrão ODMG define os seguintes elementos: *Interface* (representa o comportamento do objeto), Classe (representa o estado e o comportamento do objeto), tipos estruturados, atributos e relacionamentos. Os três últimos são responsáveis pela definição do estado do objeto dentro da classe.

As classes e *interfaces* são responsáveis por definirem o tipo do objeto e podem ser organizadas em uma hierarquia, permitindo a herança entre classes, como também entre *interfaces*. Uma classe pode conter um conjunto de instâncias, que são os objetos acessíveis e que podem ser manipulados pelo usuário, seja para, inclusão, atualização ou exclusão. O conjunto de instâncias da classe é identificado pela palavra *extent* na definição ODL da classe, como pode ser observado no exemplo da figura 3.6.

```
class Curso
  ( extent cursos
     key cod_curso )  {
  attribute short cod_curso;
  attribute string nom_curso;
  attribute short carga_horaria;
  attribute set<date> datas_reconhecimento;
}
```

Figura 3.6: Definição ODL de uma classe

Os tipos estruturados, também conhecidos como tipos definidos pelo usuário, são estruturas formadas por atributos com a finalidade de representar um domínio específico. O exemplo da figura 3.7 mostra a definição de um tipo estruturado, que no padrão ODMG é representado, em ODL, pela palavara *struct.*

```
struct Endereco {
    string rua;
    string bairro:
    string bairro:
}
```

Figura 3.7: Definição ODL de um tipo estruturado

Atributos são propriedades das classes responsáveis por definirem o estado de um objeto de uma classe.

```
class Pessoa {
  attribute string nome;
  attribute char cpf;
  attribute date data_nasc;
}
```

Figura 3.8: Definição ODL dos atributos de uma classe

O exemplo da figura 3.8 expõe a definição, em ODL, da classe `Pessoa` com os atributos `nome`, `cpf` e `data_nasc`. O padrão ODMG dispõe de um conjunto de literais que são utilizadas para especificar o domínio dos atributos. A tabela 3.1 apresenta estes literais.

Descrição	Tipos
Números inteiros	long, long long, short, unsigned long, unsigned short
Números reais	float, doublé
Lógico	Boolean
Alfanuméricos	char, string
Byte	Octet
Enumerado	Enum
Coleção sem repetição	set
Coleção com repetição	Bag
Listas	list
Vetores	Array
Data (somente o dia)	Date
Data (dia com hora)	Time
Intervalo	Interval

Tabela 3.1: Tipos de literais disponíveis no ODMG

Além dos tipos básicos apresentados na tabela 3.1, um atributo pode ter seu domínio definido por um tipo estruturado.

O relacionamento entre objetos, juntamente com os atributos, definem o estado de um objeto. O relacionamento consiste na definição da relação existente entre dois objetos, e é definido no padrão ODMG, em ODL, através da palavra *relationship* como mostra o exemplo da figura 3.9.

```
class Disciplina {
    relationship set<Professor> é_ensinada_por   inverse Professor::ensina;
}
class Professor extends Pessoa {
  relationship set<Disciplina> ensina   inverse Disciplina:: é_ensinada_por ;
}
```

Figura 3.9 : Definição ODL de relacionamentos

No exemplo da figura 3.9 é definido o relacionamento entre as classes `Disciplina` e `Professor` onde uma disciplina `é_ensinada_por` vários professores e um professor `ensina` várias disciplinas. No caso do relacionamento `ensina`, uma instância da classe `Professor` é associada a um conjunto de instâncias da classe `Disciplina`, o mesmo acontece com o relacionamento `é_ensinada_por`, onde uma instância da classe `Disciplina` é associada a um conjunto de instâncias da classe `Professor`. A representação do relacionamento inverso, indicado pela palavra *inverse*, tem como objetivo facilitar a navegação entre os diversos relacionamentos que podem ocorrer em um modelo de dados.

OIFXML

OIFXML é uma linguagem baseada em XML, idealizada por [BIE 00] que define um mapeamento para os dados organizados em um modelo OO, permitindo a sua utilização na construção de SIBDs que suportem SGBD OO. Segundo [BIE 00], OIFXML é a OIF, com a diferença que desfruta de todas as vantagens oferecidas pela XML: poder na representação da informação, facilidade de identificação dos elementos e transportabilidade.

Representando Classes e Atributos

```
<odmg_object oid=”1”>
  <class>Cliente</class>
  <contents>
    <attribute name=”Codigo”>
      <value>
         <short val=”354676”/>
      </value>
    </attribute>
    <attribute name=”PrimeiroNome”>
      <value>
         <string val=”José”/>
      </value>
    </attribute>
    <attribute name=”SegundoNome”>
      <value>
         <string val=”Silva”/>
      </value>
    </attribute>
    <attribute name=”Endereco”>
      <value>
         <string val=”Rua A n. 36”/>
      </value>
    </attribute>
  </contents>
 </odmg_object>
```

Figura 3.10: Representando classes e atributos em OIFXML

O documento XML da figura 3.10 representa a utilização da OIFXML para uma instância de uma classe *Cliente.* O marcador `<Class>` indica a classe, enquanto que `<Attribute>` indica um atributo da classe. Um fator importante desta proposta consiste na representação dos meta-dados. Tipos de dados (*string, short, boolean,* etc) são representados por uma marcação com o seu respectivo nome. No exemplo acima a marcação `<string>` indica que a informação do atributo é um conjunto de caracteres, já a marcação `<short>` representa o atributo do tipo inteiro.

Representando Atributos de Tipos Estruturados

```
<odmg_object oid=”1”>
  <class>Cliente</class>
  <contents>
    ...
    <attribute name=”Endereco”>
      <value>
        <struct>
          <field name=”rua”>
            <value>
              <string val=”Rua A”/>
            </value>
          </field>
          <field name=”numero”>
            <value>
              <string val=”36”/>
            </value>
          </field>
          <field name=”bairro”>
            <value>
              <string val=”Suissa”/>
            </value>
          </field>
        </struct>
      </value>
    </attribute>
  </contents>
 </odmg_object>
```

Figura 3.11: Representando tipos estruturados em OIFXML

Conforme a especificação ODMG, um atributo pode ser formado por uma determinada estrutura. No exemplo da figura 3.11, o atributo `Endereco` é formado pelos campos `Rua`, `Numero` e `Bairro`. As marcações `<struct>` e `<field>` permitem representar toda esta estrutura do atributo.

Representando Coleções

```
<odmg_object oid=”1”>
  <class>Cliente</class>
  <contents>
     ...
     <attribute name=”Telefones”>
        <value>
           <collection type=”set”>
              <value>
                 <string val=”9991-5090”/>
              </value>
              <value>
                 <string val=”2224-4080”/>
              </value>
           </collection>
         </value>
     </attribute>
  </contents>
</odmg_object>
```

Figura 3.12: Representando coleções em OIFXML

A utilização da marcação `<collection>` permite que atributos do tipo coleção possam ser representados. No XML da figura 3.12 , o atributo `Telefones` é uma coleção de valores do tipo *string,* estes valores são representados pelas ocorrências da marcação `<string>` dentro da marcação que representa atributo.

Representando Relacionamentos

```
<odmg_object oid=”10”>
  <class>Venda</class>
  <content>
    ...
  </content>
</odmg_object>

<odmg_object oid=”1”>
  <class>Cliente</class>
  <contents>
     ...
     <relationship name=”FizeramVendas”>
       <link to=”10”/>
     </relationship>
  </contents>
 </odmg_object>
```

Figura 3.13: Representando coleções em OIFXML

Com a marcação `<relationship>`, pode-se indicar um relacionamento de uma instância da classe `Cliente` com uma instância da classe `Vendas`, como mostra o exemplo da figura 3.13. O atributo `name` indica o nome do relacionamento e o atributo `to` da marcação `<link>` indica o `oid` de uma instância da classe relacionada. No exemplo da figura 3.13, o cliente fez a venda representada pela instância da classe `Venda` que possui o `oid` igual a 10.

Optar por OIFXML para representar os dados e esquema de um modelo OO, com o propósito de utilizar estes documentos para o intercâmbio de dados entre SGBDs, possui vantagens em relação a utilização da OIF. A figura 3.14 apresenta um exemplo da linguagem OIF na qual podem ser observadas as vantagnes discriminadas na tabela 3.2.

ODL

```
Struct Pontos {
  float X;
  float Y;
};

class Poligono {
  attribute array<Pontos> PontosRef;
};
```

OIF

```
P1 Poligono{ PontosRef { [1] X 7.5, Y 12.0}, [2] {X 22.5 Y 23.0 } } }
```

OIFXML

```
<odmg_object oid=”P1”>
  <class>Poligono</class>
  <contents>
     <attribute name=”PontosRef”>
        <value>
           <collection type=”array”>
              <value>
                <struct>
                  <field name=”X”>
                    <value>
                      <float val=”7.5”/>
                    </value>
                  </field>
                  <field name=”Y”>
                    <value>
                      <float val=”12.0”/>
                    </value>
                  </field>
                </struct>
              </value>
              <value>
                <struct>
                  <field name=”X”>
                    <value>
                      <float val=”22.5”/>
                    </value>
                  </field>
                  <field name=”Y”>
                    <value>
                      <float val=”23.0”/>
                    </value>
                  </field>
                </struct>
              </value>
           </collection>
        </value>
     </attribute>
  </contents>
</odmg_object>
```

Figura 3.14: Definição das instâncias de uma classe em OIF e OIFXML

No exemplo da figura 3.14 são representadas duas instâncias da classe Poligono, em OIF e OIFXML, chamadas de P1 e P2. É possível observer que a representação das instâncias desta classe em OIF não dispõe da mesma legibilidade da OIFXML. As marcações XML permititem que as informações sejam melhor identificadas, facilitando o entendimento do seu conteúdo.

Propriedade	**OIFXML**	**OIF**
Simplicidade	Representar esquema e dados em OIFXML, é simples uma vez que seus elementos estão bem organizados e definidos através das marcações.	Sua sintaxe para definição dos dados e do esquema é bastante complexa.
Legibilidade	O fato de OIFXML ser baseada na linguagem XML permite oferecer um alto grau de legibilidade em virtude da estruturação das marcações, fazendo com que os documentos se tornem alto-explicativos. Um ser-humano não precisa saber previamente OIFXML para entender o seu conteúdo.	A organização dos elementos não oferece uma boa legibilidade. Para um ser-humano ser capaz de entender um documento OIF é necessário que se tenha conhecimento prévio sobre a linguagem.
Padrão	A leitura de documento OIFXML pode ser feita utilizando programas padronizados que interpretam documentos XML (*XML parse*) permitindo que qualquer aplicação faça uso dos dados presentes neste documento.	Por não ser definido sobre um padrão de representação de dados, como o XML, é necessário desenvolver interpretadores específicos para utilizar seus dados.

Tabela 3.2: Comparativo entre OIFXML e OIF

3.5 Análise dos SIBDs XML

As seguintes características devem ser observadas na análise de um SIBD XML, verificando o atendimento dos requisitos apresentados na seção 3.2:

Mapeamento do Esquema e dos Dados

Deve ser verificado se o mapeamento realizado pelo SIBD atende ao segundo requisito da seção 3.2.

Transportabilidade

O nível de transportabilidade do SIBD deve ser avaliado. Mais precisamente, deve ser observada a tecnologia empregada na construção do SIBD, como a linguagem de desenvolvimento e a tecnologia de comunicação com o SGBD. Neste ponto pode-se destacar a tecnologia Java, pois, além de dar a transportabilidade necessária para este tipo de ferramenta, oferece uma API comum de acesso a banco de dados chamada de JDBC ("**J**ava **D**ata**B**ase **C**onnectivity") [SUN 99]. JDBC é uma API definida pela *Sun*[18] que padroniza o acesso a banco de dados. Para um SGBD disponibilizar acesso para aplicações escritas em Java, é necessário que a API JDBC seja implementada para este SGBD. Cada implementação da API SGBD recebe o nome de *driver* JDBC. Por exemplo, para acessar o SGBD *Oracle* a partir de uma aplicação Java é necessário utilizar um *driver* JDBC implementado para o SGBD *Oracle*.

API para Desenvolvimento

É importante que o SIBD ofereça uma API que possibilite aos programadores desenvolver aplicações que necessitem realizar, de alguma forma, importações ou exportações de informações de um banco de dados.

Resolução de conflitos

O SIBD pode em um determinado momento se deparar com esquemas de bancos de dados (origem e destino) que representam a mesma informação mas que possuem estruturas diferentes (estas diferenças são chamadas de conflitos), desta forma ele deve ser capaz de resolver estas diferenças de forma a não inviabilizar toda operação de intercâmbio de dados.

3.5.1 Ferramentas existentes para o Intercâmbio de Dados entre SGBDs

XML-DBMS

XML-DBMS é um conjunto de classes Java, que permite aos programadores construirem aplicações capazes de realizar transferência de dados entre SGBDs relacionais, utilizando XML para estruturar os dados [XML 00].

XML-DBMS realiza o intercâmbio de dados através da utilização de dois documentos XML: o primeiro é constituído dos dados propriamente ditos e o outro documento contém o mapeamento dos dados para o SGBD, especificando nomes de tabelas, de colunas e os relacionamentos entre as tabelas. A utilização de um documento de mapeamento oferece um maior detalhamento dos dados e flexibilidade na configuração do relacionamento dos dados do documento XML com os do banco de dados.

XML-DBMS é compatível com qualquer SGBD que tenha disponível *drivers* JDBC (*Oracle*. *Microsoft SQL Server, My SQL* entre outros).

Vantagens

- possui um bom mapeamento dos dados, utilizando um documento específico para esta tarefa;
- permite que documentos de dados sejam gerados a partir de sentenças SQL;
- transportabilidade – por ser desenvolvido em Java e utilizar JDBC, possibilita sua utilização em diversos ambientes (*Windows*, *Linux*, *Solaris* entre outros);

[18] Empresa criadora da linguagem Java

Desvantagens

- suporta apenas SGBD relacionais;
- não possui uma *interface* gráfica;
- não oferece tratamento quanto à exibição dos dados do documento XML;
- não realiza o mapeamento dos meta-dados;
- não realiza o tratamento de conflitos entre os banco de dados envolvidos na transação;

DB2XML

É uma ferramenta escrita em Java com código fonte aberto, onde suas classes podem ser utilizadas para extrair informações de um SGBD relacional para um documento XML. Além de extrair os dados, o DB2XML também gera um outro documento XML que irá conter os meta-dados das informações extraídas [TAU 01]. Outra característica importante desta ferramenta é a possibilidade de realizar o tratamento dos dados exportados para exibição como documento HTML. É compatível com qualquer SGBD relacional que possua drivers JDBC implementados.

Vantagens

- transportabilidade – aplicação desenvolvida em Java permite sua utilização em diversos ambientes;
- *interface* gráfica para facilitar a operação de transferência;
- tratamento dos dados para exibição;
- mapeamento dos meta-dados.

Desvantagens

- trabalha apenas com SGBD relacionais;
- opera apenas no sentido BD $\rightarrow$ XML.

XSU (XML-SQL Utility)

Esta ferramenta foi desenvolvida para realizar intercâmbio de dados entre SGBD *Oracle,* distribuído nas versões *Oracle* 8.1.7 e superiores e no *Oracle* 9i, através de XML. É capaz de atender tanto modelos relacionais quanto a objeto-relacionais suportando todos o tipos definidos no *Oracle 9i* [XSU 01].

O XSU disponibiliza três formas de utilização:

1. rodando no próprio SGBD (camada servidor) através de pacotes Java e PL/SQL[19];
2. rodando na camada de aplicação (*Application Server)* através de classes Java;
3. rodando na camada cliente (*Front-End)* através de aplicações que utilizem as classes Java para o intercâmbio ou utilizando aplicações já prontas disponibilizadas pelo XSU.

Quando o XSU atua na camada de aplicação ou na cliente, sua conexão com o SGBD é realizada através de *drivers* JDBC. Mesmo utilizando JDBC e XML, componentes importantes no processo de intercâmbio de dados, o XSU não garante o funcionamento para SGBDs que não sejam *Oracle*.

Uma funcionalidade importante do XSU é a possibilidade de fazer não só inserção de informações no banco de dados a partir de um XML, como também a possibilidade de atualizações e exclusões de dados.

Vantagens

- suporta modelos relacionais e objeto-relacionais;
- realiza inserções, atualizações e exclusões em um banco de dados;
- dispõe de mecanismos para o processamento nas camadas de servidor, aplicação e cliente.

Desvantagens

- garante apenas o funcionamento para SGBD *Oracle*;
- não faz o mapeamento dos meta-dados;
- não dispõe de mecanismos para o tratamento dos dados para exibição;

[19] Linguagem para desenvolvimento no SGBD Oracle

- Ø não trata possíveis conflitos existentes entres os banco de dados de origem e destino;
- Ø não oferece uma *interface* gráfica.

XLE (XML Lightweight Extractor)

Ferramenta desenvolvida pela *AlphaWorks*[20] para a extração de dados de tabelas relacionais de um determinado SGBD através de conexão JDBC. A construção do XML é definida pelo usuário através de um documento chamado DTDSA, onde estão definidos as tabelas e os relacionamentos entre elas e alguns parâmetros para a execução do XLE [XLE 99].

Vantagens

- Ø o documento XML é produzido de acordo com a especificação do usuário (utilizando o documento DTDSA);
- Ø transportabilidade – pode ser utilizado em diversas plataformas (*Windows, Solaris, Linux, Unix, AS/400*) e em SGBD que utilizem conexão via JDBC.

Desvantagens

- Ø trabalha apenas com SGBD relacionais;
- Ø opera apenas no sentido BD → XML;
- Ø não realiza o mapeamento dos meta-dados;
- Ø não dispõe de mecanismo para o tratamento dos dados para exibição;
- Ø não oferece *interface* gráfica.

3.6 Conclusão

Do conteúdo apresentado neste capítulo podemos chegar a conclusão que, para o SIBD ser uma aplicação capaz de realizar o intercâmbio de dados entre diferentes SGBDs, é necessário que seja capaz de estruturar as informações provenientes de uma base de dados em

[20] http://www.alphaworks.ibm.com/

documentos XML, fazendo com que estes sejam utilizados como “meio de transporte de dados entre os SGBDs”, de oferecer suporte a diferentes SGBDs bem como a diferentes plataformas e que disponibilize uma API para o desenvolvimento de aplicações.

Na avaliação dos SIBDs existentes, pode ser observado uma carência em relação ao suporte de diferentes tipos de SGBDs. Nesta avaliação, o SGBD relacional aparece como o principal tipo de SGBD suportado, outros tipos, como o objeto-relacional e orientado a objeto quase não são atendidos, limitando o universo de abrangência destas ferramentas. Outra deficiência destes SIBDs está no fato de não oferecer tratamento as possíveis diferenças (conflitos) que geralmente existem entre os banco de dados de origem e destino. A tabela 3.3 apresenta um relação entre os SIBDs descritos neste capítulo e os requisitos que devem ser atendidos, detalhados na seção 3.2, como também as dificiências observadas na análise feita na seção 3.7.1 de forma a se ter um visão geral das limitações existentes nestes produtos.

Funcionalidade	**XML-DBMS**	**DB2XML**	**XSU**	**XLE**
Estruturar os dados em um documento padrão	Sim	Sim	Sim	Sim
Suporte a diferentes SGBDs	Sim	Sim	Não	Sim
Suporte a diferentes plataformas	Sim	Sim	Sim	Sim
Seleção de dados para importação/exportação	Sim	Sim	Sim	Sim
API para desenvolvimento de aplicações	Sim	Sim	Sim	Sim
Suporte àresolução de conflitos	Não	Não	Não	Não
Suporte a SGBDs OR	Não	Não	Sim	Não
Suporte a SGBDs OO	Não	Não	Não	Não
Interface Gráfica	Não	Sim	Não	Não
Tratamento de dados para exibição	Não	Sim	Não	Não

Tabela 3.3: **Relação SIBDs x Funcionalidades**

A proposta deste trabalho é justamente suprir estas limitações, oferecendo um SIBD, XML-ODBMS, capaz de suportar SGBD OO e SGBD OR segundo um modelo orientado a objeto.

Capítulo 4

XML-ODMG

4.1 Introdução

A fim de prover uma boa estruturação dos dados que serão intercambiados entre SGBDs (OO, OR ou R), desenvolvemos um SIBD – XML-ODBMS – que oferece uma linguagem – XML-ODMG – para a definição das regras de mapeamento do esquema e dos dados de um banco de dados para um documento estruturado intermediário, conforme o esquema básico de funcionamento de um SIBD apresentado na figura 3.1.

XML-ODMG é uma linguagem baseada em XML e compatível com o padrão ODMG [CAT 00]. Permite a representação de esquemas e dados de bancos de dados orientados a objeto ou objeto-relacionais (classes, relacionamentos, atributos, etc) como também de esquemas relacionais e objeto-relacionais mediante um mapeamento dos elementos de cada modelo para os elementos definidos no ODMG, e ainda contempla a definição de regras para a resolução de conflitos existentes entre os banco de dados origem e destino. A resolução de conflitos é um requisito fundamental, pois é comum situações em que o SIBD se depara com esquemas semelhantes, mas que mesmo assim apresentem diferenças, sejam estruturais ou semânticas [SIL 01]. O modelo de dados implementado pela XML-ODMG é muito próximo do modelo de dados semiestruturado apresentado em [ABI 00].

A linguagem XML-ODMG é definida por meio de três documentos: XML-ODMG de Esquema, XML-ODMG de Dados e XML-ODMG de Transformação e de Resolução de Conflitos.

Neste capítulo, apresentaremos as definições dos três tipos de documentos da XML-ODMG, ilustradas com exemplos.

4.2 A Linguagem XML-ODMG

A linguagem XML-ODMG está descrita em três documentos:

1. **XML-ODMG de Esquema:** contém informações sobre *interfaces*, classes, atributos das classes, relacionamentos entre as classes, herança de classes, tipos de atributos, enfim todos os elementos que compõem um banco de dados orientado a objeto de onde os dados devem ser intercambiados com outros bancos de dados — banco de dados origem. Este documento também pode ser utilizado para representar esquemas objeto-relacionais e relacionais através do mapeamento destes dois modelos para os elementos do modelo orientado a objeto.
2. **XML-ODMG de Dados:** trata dos dados propriamente ditos, organizados conforme o esquema definido no primeiro documento.
3. **XML-ODMG de Transformação e Resolução de Conflitos:** este documento é utilizado no momento da importação dos dados descritos no documento XML-ODMG de Dados para o banco de dados destino. Contém o esquema do banco de destino e a especificação da resolução de possíveis conflitos entre os bancos origem e destino.

4.2.1 A Linguagem XML-ODMG de Esquema

O documento XML-ODMG de Esquema é a descrição, na sintaxe XML, do esquema ODL/ODMG de um banco de dados.

Os elementos da XML-ODMG de Esquema serão ilustrados com o exemplo das figuras 4.1 e 4.2. O DTD para a definição de um documento XML-ODMG de Esquema é o assunto do apêndice A, enquanto o documento XML-ODMG completo do exemplo ilustrativo é apresentado no apêndice B.

```
class Curso
  ( extent cursos
    key cod_curso )  {
  attribute short cod_curso;
  attribute string nom_curso;
  attribute short carga_horaria;
  attribute set<date> datas_reconhecimento;
```

```
Struct Endereco {
   String numero;
   String rua;
   String bairro;
   String cidade;
   Char estado;
}
```

```
  relationship set<Aluno> é_cursado_por
    inverse Aluno::cursa;
}

struct Desconto {
  char tipo_desconto;
  float valor;
}

interface Aluno-IF {
  attribute short matricula;
  attribure float media;
  attribute Desconto desconto;
  attribute set<
      struct Matricula {
           float nota,
           short faltas,
           relationship set<Disciplina> faz
              inverse Disciplina::é_feita_por } >
       matriculas;
  relationship Curso cursa
      inverse Curso::é_cursado_por;
   void matricular( in short cod_disc );
   void cancelar_matricula( in short cod_disc );
   void transferir_matricula (
                    in short cod_disc_velho,
                    in short cod_disc_novo );
}

class Disciplina
  ( extent disciplinas,
    key cod_disc  ) {
  attribute short cod_disc;
  attribute string nom_disc;
  attriburte short carga_horaria;
  relationship set<Aluno> é_feita_por
     inverse Aluno::faz;
  relationship set<Professor> é_ensinada_por
     inverse Professor::ensina;
}

struct Titulacao {
  attribute string titulo;
  attribute string instituicao;
  attribute date inicio;
  attribute date fim;
}

class Pessoa {
  attribute string nome;
  attribute char cpf;
  attribute date data_nasc;
  attribute Endereco endereco;
  attribute set<string> telefones;
}

class  Aluno extends Pessoa : Aluno-IF
    ( extent alunos
       key matricula ) {
  attribute short matricula;
  attribure float media;
  attribute string reprensao;
  attribute Desconto desconto;
  attribute set<
     struct Matricula {
          float nota,
          short faltas,
          relationship set<Disciplina> faz
             inverse Disciplina::é_feita_por } >
      matriculas;
  relationship Curso cursa
     inverse Curso::é_cursado_por;
 }

class Centro
    (extent centros
     key cod_centro) {
  attribute short cod_centro;
  attribute string nom_centro;
  relationship set<Professor> possui
      inverse faz_parte :: Professor;
}

class Professor extends Pessoa
  ( extent professores
    key cod_prof ) {
  attribute short cod_prof;
  attribute set<Titulacao> titulacoes;
  relationship set<Disciplina> ensina
     inverse é_ensinada_por :: Disciplina
  relationship Centro faz_parte
     inverse possui :: Centro;
}
```

Figura 4.1: Esquema ODL analítico

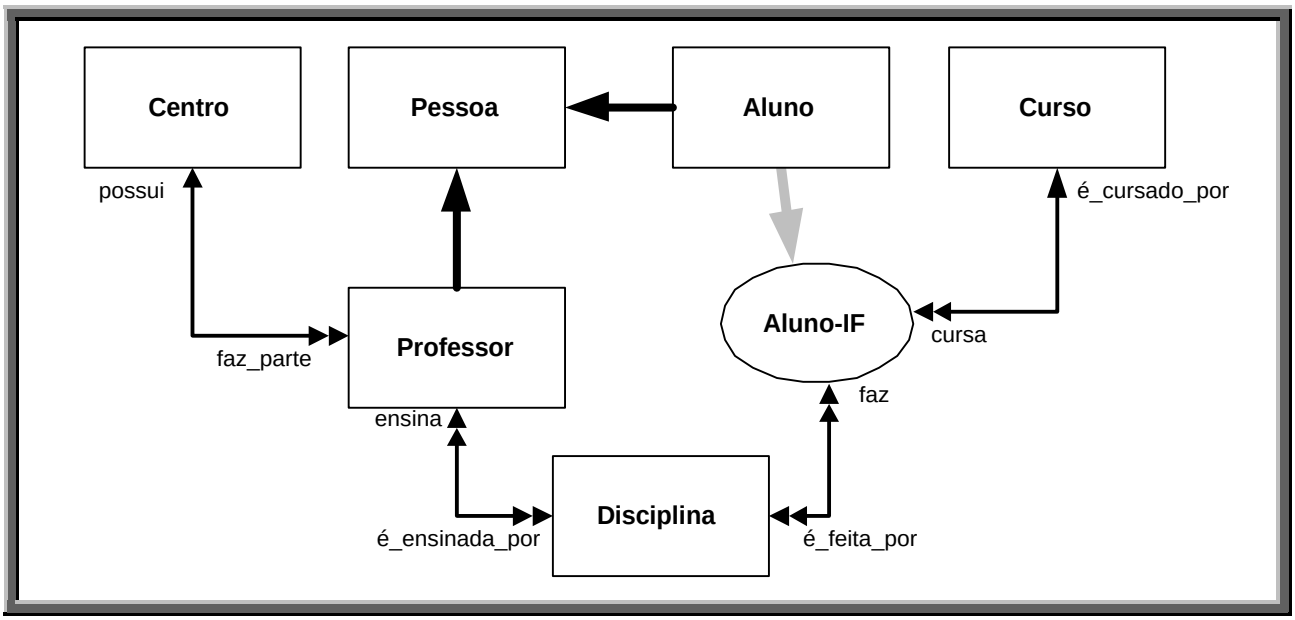

Figura 4.2: Esquema ODL gráfico

Elemento XML-ODMG

É a "raiz" da hierarquia do documento XML-ODMG de Esquema que conterá outros elementos que representarão as classes, as *interfaces*, os atributos, os tipos estruturados entre outros (figura 4.3, em azul).

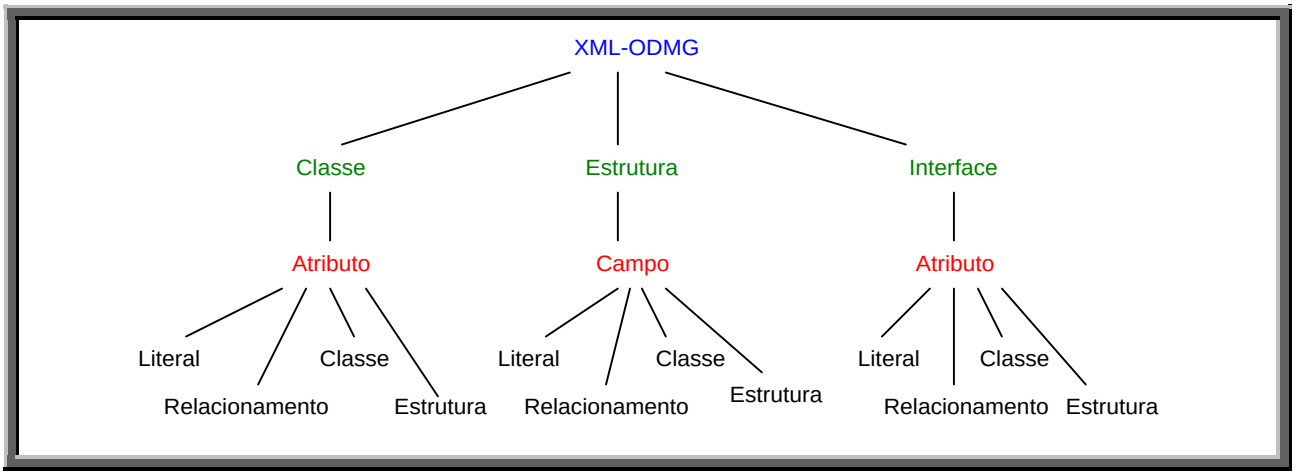

Figura 4.3: Hierarquia de elementos da XML-ODMG de Esquema

Este elemento contém atributos com informações referentes ao SGBD utilizado, para conexão com o mesmo e formatação de dados (ver tabela 4.3 e figura 4.4).

Atributo	Opcional	Valor *Default*	Valores	Descrição
typeDB	Não		OBJECT-RELATIONAL ORIENTED-OBJECT RELATIONAL	Indica o tipo do banco de dados em que se está trabalhando, podendo ser objeto-relacional, orientado a objeto ou relacional.
urlJDBC	Não			Informa a *string* JDBC de conexão para o SGBD.
username	Não			Indica o nome do usuário para conexão com o SGBD.
password	Não			Informa a senha de conexão com o SGBD.
ODBMS	Não			Indica o SGBD que se está trabalhando (Oracle, Jasmine, PostgreSQL, etc).
dateFormat				Indica o formato da data que será trabalhado.
timeStampFormat				Indica o formato de um valor *Timestamp*[21].

Tabela 4.1: Atributos do elemento *XML-ODMG*

```
<xml-odmg typeDB=”OBJECT-RELATIONAL” urlJDBC=”jdbc:oracle:thin:@localhost:orcl”
        username=”xml” password=”xml ” odbms = “oracle” dateFormat=”dd/MM/yyyy”
        timeStampFormat=”HH:mm:ss”>
   <!- - Definição das marcações que representam os elementos dos bancos de dados - ->
</xml-odmg>
```

Figura 4.4: Definição do elemento XML-ODMG para o documento de esquema

Representando Classes

A representação das classes de um modelo orientado a objeto é feita através da utilização do elemento `class`. A tabela 4.2 apresenta os atributos que formam este elemento e a figura 4.5 um exemplo desta representação.

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Indica o nome conceitual da classe no modelo de dados
idt	Não			É um identificador único para classe. Este identificador permite que outros elementos do documento façam referência a esta classe
nameDB	Não			Indica o nome da classe no banco de dados.
extent	Sim			Representa o nome das instâncias da classe persistentes no banco de dados. Para as classes que não são instanciadas, este atributo não é necessário.

Tabela 4.2: Definição dos atributos do elemento `class`

```
<class name=”Curso” idt=”curso” nameDB=”T_CURSO” extent=”CURSOS“>
  <!- - Definição dos atributos, relacionamentos e herança - ->
</class>
```

Figura 4.5: Representação classes em XML-ODMG

A representação dos métodos de uma determinada classe não se faz necessária, pois a finalidade da linguagem XML-ODMG é prover uma estrutura de documento para o intercâmbio dos dados, informações sobre o comportamento destes dados (métodos) não são contundentes para este tipo de operação[GRA 02].

Representando *Interface*

Uma *interface* do modelo de dados é representada através do uso da marcação *interface*. A tabela 4.3 mostra os atributos que devem ser utilizados nesta marcação e a figura 4.6 um exemplo da representação de uma *interface* em XML-ODMG .

[21] Tipo de dado que contém data e hora conjugados

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Indica o nome conceitual da interface no modelo de dados
Idt	Não			É um identificador único para interface. Através deste identificador possibilita que outros elementos do documento façam referência a esta interface.
nameDB	Não			Indica o nome da interface no banco de dados.

Tabela 4.3: Definição dos atributos do elemento `interface`

```
<interface name=”Aluno-IF” idt=”aluno-if” nameDB=”ALUNO_IF”>
  <!- - Definição dos atributos, relacionamentos e herança - ->
</inteface>
```

Figura 4.6: Representação de *interfaces* em XML-ODMG

Diferentemente da representação da classe o elemento `interface` não possui o atributo `extents`, uma vez que uma *interface*, por definição, não pode possuir instâncias [CAT 00].

O atributo `idt` é um identificador único para cada elemento representado, ou seja, este valor não pode se repetir tanto para classes quanto para *interfaces*.

Representando um tipo estruturado

Para representar tipos estruturados definidos pelo usuário, deve-se utilizar a marcação `struct`. A tabela 4.4 apresenta os atributos que devem compor esta marcação.

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Indica o nome conceitual da estrutura definida
Idt	Não			É um identificador único para o tipo estruturado. Através deste identificador pode-se definir atributos deste tipo.
nameDB	Não			Indica o nome do tipo estruturado no banco de dados.

Tabela 4.4: Definição dos atributos do elemento `struct`

O fragmento de um documento XML da figura 4.7 apresenta a representação de um tipo estruturado em XML-ODMG.

```
<struct name=”Endereco” idt=”endereco” nameDB=”T_ENDERECO”>
  <!- - Definição dos atributos e relacionamentos - ->
</struct>
```

Figura 4.7 : Representação de tipos estruturados em XML-ODMG

O atributo `idt` deste elemento também não pode ter o mesmo valor de um outro tipo estruturado, classe ou *interface,* ou seja, deve ser único dentro do documento.

Representando uma herança entre classes

Para representar a herança entre classes, a linguagem XML-ODMG define um elemento de nome `objectType`, que indica a classe pai da classe que está sendo representada.

```
<class name=”Pessoa” idt=”pessoa” nameDB=”T_PESSOA”>
  <!- - Definição dos atributos, relacionamentos e herança - ->
</class>

<class name=”Professor” idt=”professor” nameDB=”T_PROFESSOR” >
        extent=”PROFESSORES“>
  <objectType ref=”pessoa”/>
  <!- - Definição dos atributos e relacionamentos - ->
</class>
```

Classe pai

Indica a herança

Figura 4.8: Representação de herança em XML-ODMG

O exemplo da figura 4.8 representa a herança entre a classe `Professor` e a classe `Pessoa`. O atributo `ref` do elemento `objectType` contém o valor do `idt` da classe `Pessoa` da qual estão sendo herdadas as características. Através deste valor de referência é possível representar a herança entre duas classes, não suportando herança múltipla.

Esta mesma representação se aplica para a herança entre interfaces ou para quando uma classe implementa uma *interface* como no exemplo da figura 4.9, que mostra a definição da classe `Aluno` que implementa a interface `Aluno-IF`.

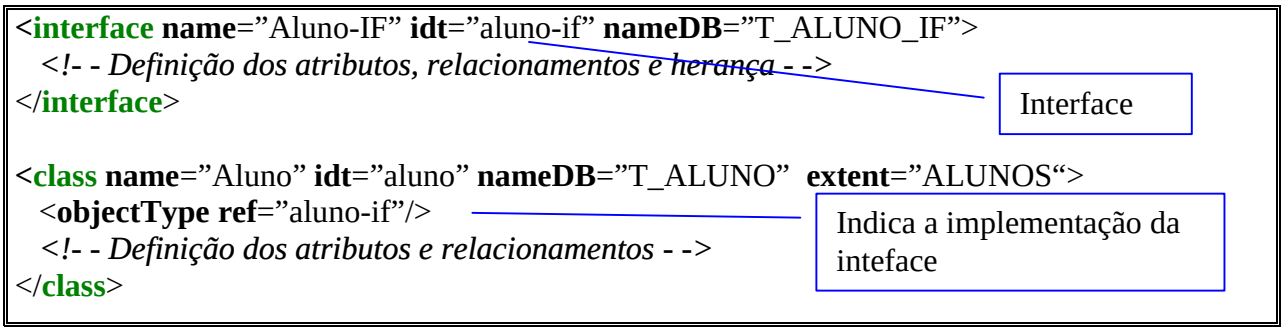

Figura 4.9: Representação da implementação de um *interface* por uma classe em XML-ODMG

Representando atributos de uma classe

A representação dos atributos de uma classe é feita através da marcação `attribute` incluída na marcação `class` que representa a classe, como no exemplo da figura 4.10. A tabela 4.5 exibe os atributos que devem ser utilizado na marcação `attribute`.

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Nome conceitual do atributo.
nameDB	Não			Nome do atributo no banco de dados.
key	Não	no	yes	Indica se o atributo faz parte da composição da chave da classe.
			no	
nullable	Não	yes	yes	Representa se o atributo pode conter valores nulos. Caso seja "yes" o atributo pode ter valor nulo, se for "no" é obrigatório o preenchimento deste atributo.
			no	
collection	Não	no	yes	Indica se atributo é formado por uma coleção. Em caso positivo, o valor deste atributo deve ser preenchido com "yes", caso contrário por "no".
			no	
collectionType	Sim		set	Indica os tipos de coleções de um atributo definidos pelo ODMG. Este valor apenas deve ser preenchido se o valor do atributo *collection* for "yes".
			bag	
			list	
			Array	
			dictionary	
type	Não	literal	literal	Representa o tipo de estrutura utilizado pelo atributo. Podendo ser uma literal ("literal"), um tipo definido pelo usuário ("struct"), um relacionamento ("relationship") ou uma classe ("class").
			relationship	
			class	
			struct	
length	Sim			Indica o tamanho de um atributo. Utilizado apenas para atributos do tipo literal.

Tabela 4.5: Definição dos atributos do elemento `attribute`

```
<class name=”Curso” idt=”curso” nameDB=”T_CURSO” extent=”CURSOS“>
   <attribute name=”cod_curso” nameDB=”COD_CURSO” key=”yes” nullable=”no”
                type=”literal” length=”4”>
      <!- - Element que representará o tipo do atributo - ->
   </attribute>
      <!- - Definição de outros atributos e relacionamentos - ->
</class>
```

Figura 4.10: Representação de atributos de uma classe em XML-ODMG

Representando os campos de um tipo estruturado

Para representar os campos (os campos estão para o tipo estruturado assim como os atributos estão para as classes) de um tipo estruturado, utiliza-se a marcação `field` que são inseridas dentro da marcação `struct`. As características deste elemento são idênticas a da marcação `attribute`, como pode ser observado na tabela 4.6. A figura 4.11 mostra um fragmento de um documento XML que define um tipo estruturado em XML-ODMG.

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Nome conceitual do campo.
nameDB	Não			Nome do campo no banco de dados.
nullable	Não	yes	yes	Representa se o campo pode conter valores nulos. Caso seja “yes”, o campo pode ter valor nulo, se for “no” é obrigatório a preenchimento deste campo.
			no	
collection	Não	no	yes	Indica se campo é formado por uma coleção. Em caso positivo, o valor deste campo deve ser preenchido com “yes”, caso contrário por “no”.
			no	
collectionType	Sim		set	Indica os tipos de coleções de um campo definidos pelo ODMG. Este valor apenas deve ser preenchido se o valor do campo *collection* for “yes”.
			bag	
			list	
			array	
			dictionary	
type	Não	literal	literal	Representa o tipo de estrutura utilizado pelo campo. Podendo ser uma literal (“literal”), um tipo definido pelo usuário (“struct”), um relacionamento (“relationship”) ou uma classe (“class”).
			relationship	
			class	
			struct	
length	Sim			Indica o tamanho de um campo. Utilizado apenas para campos do tipo literal.

Tabela 4.6: Definição dos atributos do elemento `field`

```
<struct name=”Endereco” idt=”endereco” nameDB=”T_ENDERECO”>
   <field name=”numero” nameDB=”NUMERO” type=”literal” length=”5”>
      <!- - Elemento que representará o tipo do campo - ->
   </field>
  <field name=”rua” nameDB=”RUA” type=”literal” length=”80”>
      <!- - Elemento que representará o tipo do campo - ->
   </field>
</struct>
```

Figura 4.11: Representação de campos de um tipo estruturado em XML-ODMG

Representando atributos e campos do tipo literal

Para indicar um atributo ou um campo que seja do tipo literal, faz-se uso da marcação `literal` inserida dentro das marcações `attribute` ou `struct`. Através da utilização desta marcação, é possível indicar o tipo da literal, seja *string, short, long, float,* enfim todos os tipos definidos pelo ODMG. A figura 4.12 exemplifica esta definição e a tabela 4.7 define o atributo que devem ser utilizados na marcação `literal`.

Atributo	**Opcional**	**Valor *Default***	**Valores**	**Descrição**
type	Não	string	long long_long short unsigned_long unsigned_short float double boolean octet char string time date timestamp	Indica o tipo da literal relacionada ao atributo de uma classe ou a um campo de um tipo estruturado. Os valores possíveis deste atributo estão relacionados com os tipos de literais definidos pelo ODMG.

Tabela 4.7: Definição dos atributos do elemento `literal`

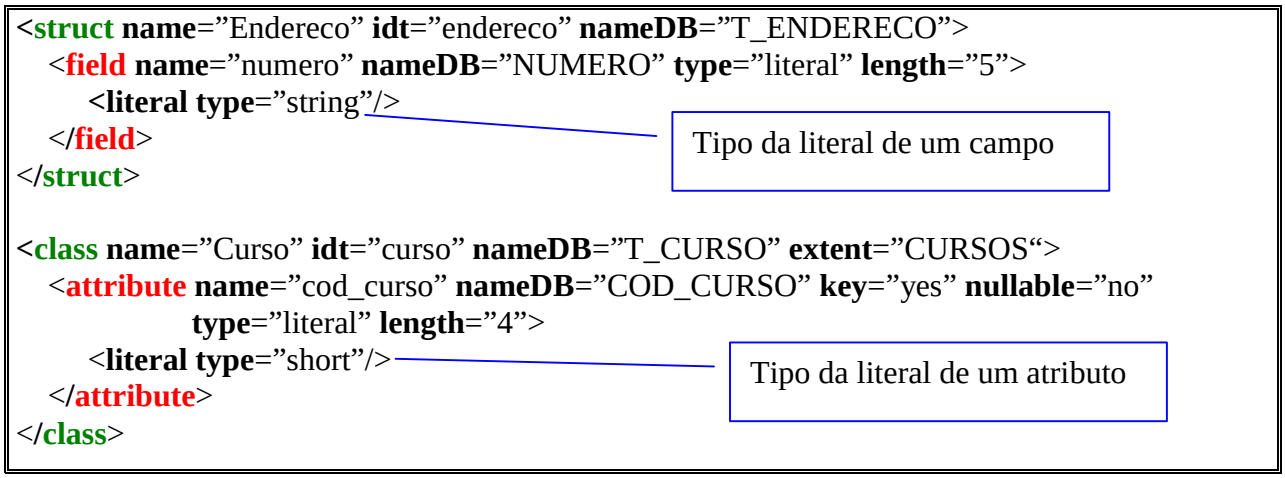

Figura 4.12: Representação de atributos e campos do tipo literal em XML-ODMG

Representando tipos estruturados para atributos e campos

Um atributo ou um campo pode ter seu domínio definido sobre um tipo estruturado. Para realizar esta representação em XML-ODMG, utiliza-se a mesma marcação da representação de herança, ou seja o elemento `objectType`.

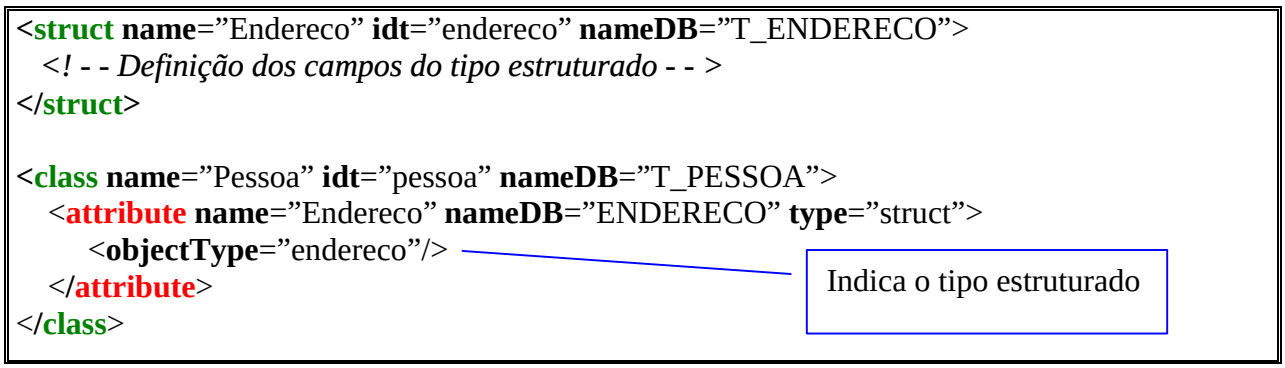

Figura 4.13: Representando atributos e campos definidos como um tipo estruturado em XML-ODMG

No exemplo da figura 4.13 o atributo `endereco` da classe `Pessoa` é de um tipo estruturado. Para esta representação o valor do atributo `ref` da marcação `objectType` deve ter o mesmo valor do `idt` do elemento que representa a estrutura `Endereco` e o valor do atributo `type` da marcação `atributo` deve ser “struct”, indicando que se está fazendo referência a um tipo estruturado. O tipo do atributo também pode ser definido por uma classe, esta representação deve ser feita da mesma maneira do tipo estruturado, ou seja, o valor do

atributo `ref` da marcação `objectType` deve ter o mesmo valor do `idt` de uma determinada classe.

Representando um relacionamento

Utilizando a marcação `relationship` inserida nas marcações de representação de atributos ou de campos é possível representar o relacionamento existente entre classes. Os atributos desta marcação são apresentados na tabela 4.8.

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Nome do relacionamento.
classRef	Não			Indica a classe de referência. O valor deste atributo deve ser igual ao *idt* da classe que se está relacionando.
relationShipRef	Não			Nome do relacionamento inverso.

Tabela 4.8: Definição dos atributos do elemento `relationship`

```
<class name=”Aluno” idt=”aluno” nameDB=”T_ALUNO” extent=”ALUNOS“>
    <objectType ref=”aluno-if”/>
   <attribute name=”curso” nameDB=”CURSO” type=”relationship”>
       <relationship name=”cursa” classRef=”curso” relationshipRef=”e_cursado_por”/>
   </attribute>
   < !- - Demais atributos - ->
</class>
```

Figura 4.14: Representação de relacionamento em XML-ODMG

A figura 4.14 é um fragmento de um documento XML e mostra a representação de um relacionamento entre classes definido em XML-ODMG, onde o atributo `type` do elemento `attribute` deve possuir o valor “relationship” a fim de indicar que o atributo se trata de um relacionamento entre classes. Os atributos `nullable`, `key` e `length` não são aplicados na representação de relacionamentos.

Representando coleções

As coleções são definidas tanto para atributos como para campos e podem ser de valores do tipo literal, de tipos estruturados ou de relacionamentos. Esta representação deve ser feita configurando os valores dos atributos `collection` e `collectionType`.

```
<class name="Curso" idt="curso" nameDB="T_CURSO" extent="CURSOS">
  <attribute name="alunos" nameDB="ALUNOS" literal="relationship"
             collection="yes" collectionType="set">
    <relationship name="e_cursado_por" classRef="aluno" relationshipRef="cursa"/>
  </attribute>
  <!- - Definição de outros atributos e relacionamentos - ->
</class>
```

Coleção de relacionamentos

Figura 4.15: Representação de coleções em XML-ODMG

O exemplo da figura 4.15 representa um atributo da classe `Curso` que possui um conjunto, do tipo *set*, de referências (ou relacionamentos) para a classe `Aluno`.

4.2.2 A Linguagem XML-ODMG de Dados

O XML-ODMG de Dados é estruturado conforme a definição do XML-ODMG de Esquema. Este irá conter as instâncias das classes definidas no modelo de dados respeitando os relacionamentos existentes entre elas e as características de cada atributo definido. A estrutura do XML-ODMG de Dados é simples, consiste em um elemento principal chamado de `xml-odmg-data` e de instâncias de uma ou mais classes vinculadas a este elemento (ver figura 4.3). Diferente do XML-ODMG de Esquema, o XML-ODMG de Dados não contém marcações pré-definidas, elas são definidas de acordo com as classes representadas no XML-ODMG de Esquema.

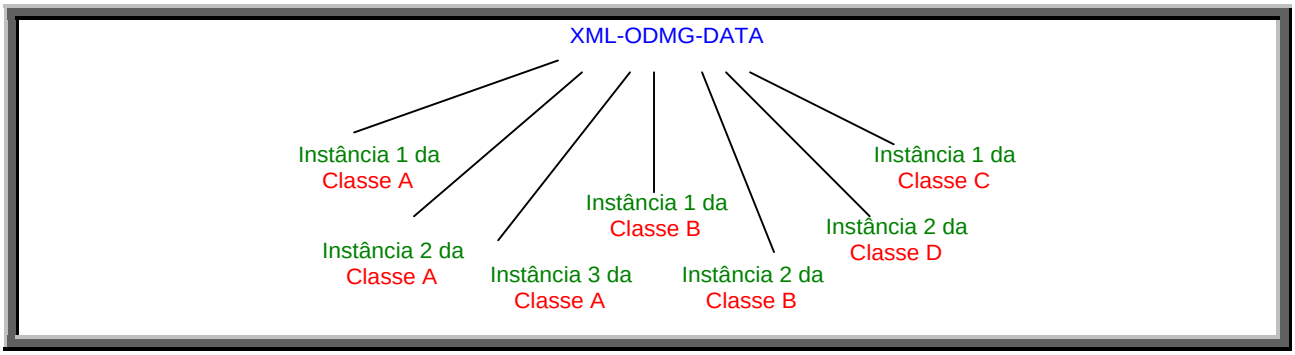

Figura 4.16: Hierarquia de elementos do XML-ODMG de Dados

Representando instâncias de uma classe

Cada instância de uma classe é representada por uma marcação que tem o seu nome, onde cada uma vai possuir atributos que representam os atributos da classe, como mostra o exemplo da figura 4.17.

```
<xml-odmg-data>
  <Curso oid=”1” cod_curso=”10” nom_curso=”Direito” carga_horaria=”780”/>
  <Curso oid=”2” cod_curso=”15” nom_curso=”Informática” carga_horaria=”780”/>
  <!- - Outras instâncias da classe curso e de outras classes - ->
</xml-odmg-data>
```

Figura 4.17 : Representação de instância de uma classe em XML-ODMG

Cada ocorrência da marcação `Curso` representará uma instância da classe `Curso` do banco de dados. Toda representação de uma instância de classe conterá o atributo `oid`. O valor deste atributo é único entre as instâncias de qualquer classe e é utilizado para permitir que instâncias façam referência a outras instâncias, a fim de projetar os relacionamentos definidos no XML-ODMG de Esquema.

Representando instâncias de uma classe que possuam atributos de um tipo estruturado

Quando um atributo ou um campo é de um tipo definido pelo usuário, uma outra marcação, com o mesmo nome do atributo ou campo, é criada dentro da marcação que representa o atributo ou campo, como pode ser observado na figura 4.18.

```
<xml-odmg-data>
  <Aluno oid=”4” matricula=”126598” media=”5,5”>
    <desconto tipo_desconto=”Bolsa” valor=”100.25”/>
  </Aluno>
</xml-odmg-data>
```

Figura 4.18: Representação de instância de um classe que possuem atributos definidos como tipo estruturado em XML-ODMG

O atributo `desconto` da classe `Aluno` é um tipo estruturado, desta forma a representação destes dados é feita através da marcação `desconto` incluída dentro da representação da instância da classe e os valores dos campos do tipo estruturado são definidos como atributos desta marcação.

Representando instâncias com relacionamentos

Para fazer o relacionamento entre instâncias no XML-ODMG de Dados, a linguagem XML-ODMG utiliza-se do atributo `oid`. Quando ocorre um relacionamento é criado um atributo no elemento da instância que vai conter o valor do `oid` da instância da classe da qual existe o relacionamento.

```
<xml-odmg-data>
  <Curso oid=”1” cod_curso=”10” nom_curso=”Direito” carga_horaria=”780”/>
  <Aluno oid=”15” matricula=”9999” media=”5.5” curso=”1”>
    <!- - Definição de outros atributos -- >
  </Aluno>
</xml-odmg-data>
```

Referência à instância com *idt* igual a “1”

Figura 4.19: Representação de instâncias de classe, que possuem relacionamento, em XML-ODMG

O exemplo da figura 4.19 mostra uma instância da classe `Aluno` fazendo referência a uma instância da classe `Curso`. O atributo `curso` do elemento `Aluno` possui o mesmo valor do `oid` de um elemento `Curso` que representa uma instância desta classe. Através da igualdade de valores entre estes dois atributos é possível indicar o curso em que um determinado aluno faz parte.

Representando coleções

As coleções podem ocorrer, em um modelo orientado a objeto, de três formas diferentes. A primeira acontece quando um atributo contém um conjunto de valores do tipo literal, a segunda quando o conjunto é formado por tipos estruturados e por último quando este é formado por um conjunto de referências (relacionamentos).

Coleção de literais

Um conjunto de literais é representado por uma coleção de elementos (onde cada elemento representará um item da coleção) com o mesmo nome do atributo, que são inseridos dentro da marcação que representa a instância da classe ou do tipo estruturado. Para cada ocorrência desta marcação existirão dois atributos, onde juntos identificarão o índice do item e valor de cada item. Estes atributos são respectivamente `idx` e `value`.

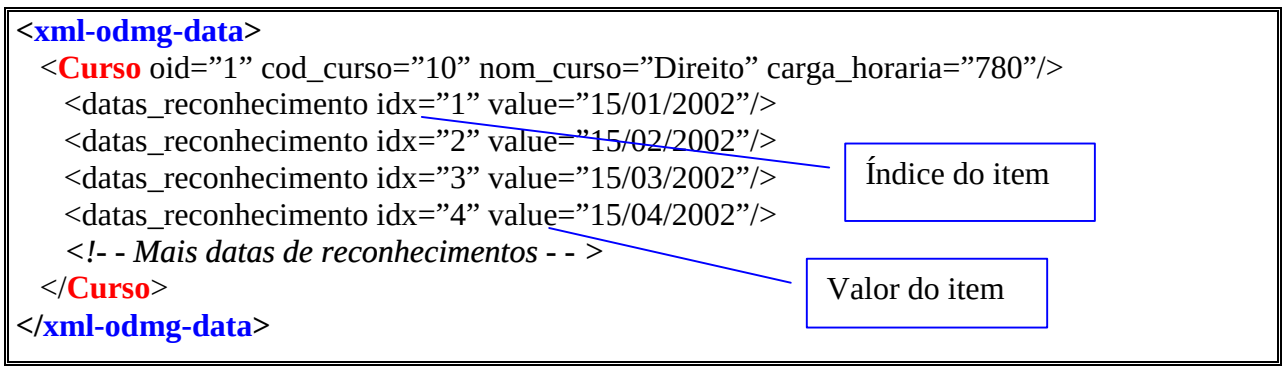

Figura 4.20 : Representação de instância de classes que possuem coleção de literais, em XML-ODMG

As marcações `datas_reconhecimento`, no exemplo da figura 4.20, indicam o conjunto de datas de reconhecimento para um determinado curso. Através do atributo `value` é possível recuperar as datas que formam o conjunto do atributo `datas_reconhecimento`.

Coleção de tipos estruturados

Para representar os dados de uma coleção de tipos estruturados, a XML-ODMG utiliza-se de um conjunto de elementos, com o mesmo nome do atributo, que ocorrem dentro

da marcação que representa a instância da classe. Cada elemento possui um atributo de nome "idx" que indicará o índice do item dentro do conjunto e um outro elemento que representará os valores dos campos do tipo estruturado, onde este possuirá o mesmo nome do tipo estruturado.

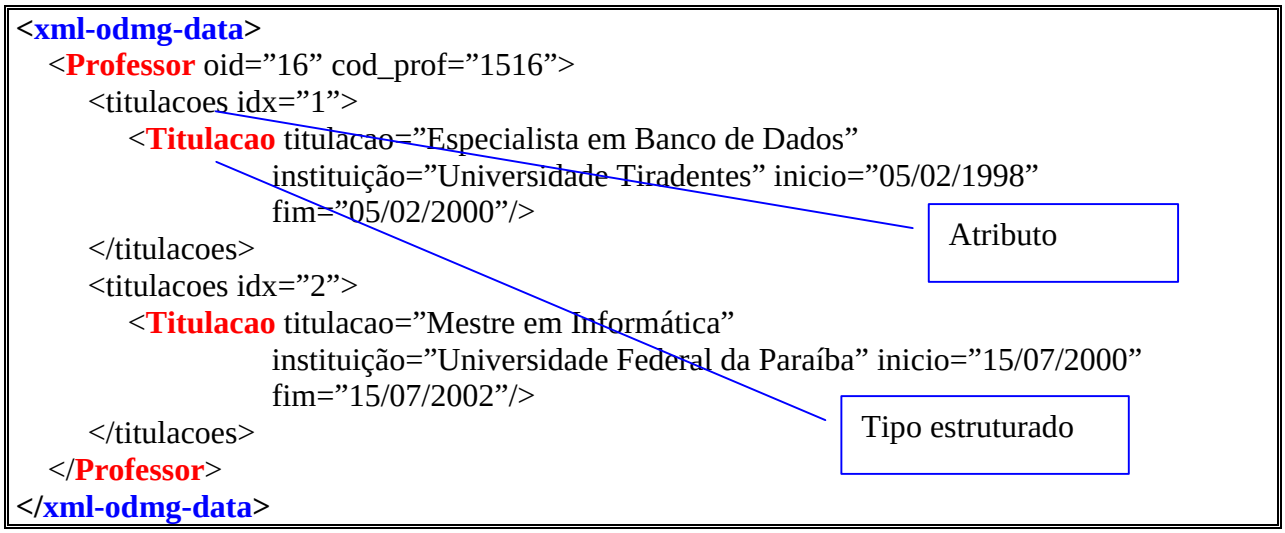

Figura 4.21: Representação de instâncias de classes que possuem coleção de tipos estruturados, em XML-ODMG

No documento XML da figura 4.21 que representa instâncias de uma classe que possui coleção de tipos estruturados, cada ocorrência da marcação `titulacoes` indicará uma titulação do professor. Estas marcações conterão um atributo, para representar o índice do item dentro do conjunto, `idx` , e uma outra marcação para representar os valores do tipo estruturado, `Titulacao`.

Coleções de referências (Relacionamentos)

A representação deste tipo de coleção é semelhante à representação da coleção de literais. A principal diferença consiste no valor do atributo `value`. Enquanto que na representação de tipos literais este atributo contém o dado proprieamente dito, na representação de referência o mesmo atributo vai conter o valor do `oid` de uma determinada instância.

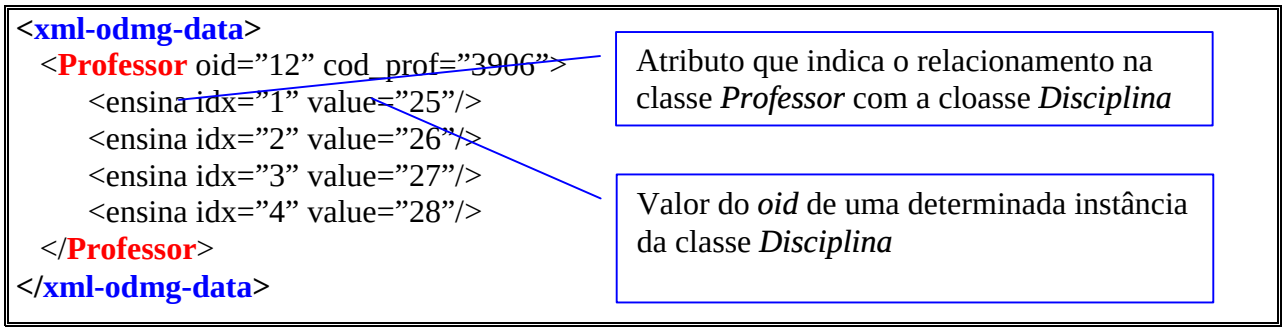

Figura 4.22: Representação de instâncias de classes que possuem uma coleção de referência (relacionamento), em XML-ODMG

Para cada ocorrência da marcação `ensina`, no exemplo da figura 4.22, o valor do atributo `value` conterá o valor do `oid` de uma determinada instância da classe `Disciplina`. Através desta representação é possível recuperar todas as disciplinas que um determinado professor ensina.

Representando herança

Para representar os dados que estão estruturados em uma classe que herda características de outra, a XML-ODMG utiliza-se do mesmo princípio para a representação de atributos ou campos que são de um tipo estruturado, ou seja, é definido um elemento dentro da marcação que representa a instância da classe, a qual vai conter os atributos da classe herdada. Esta representação pode ser observada no exemplo da figura 4.23.

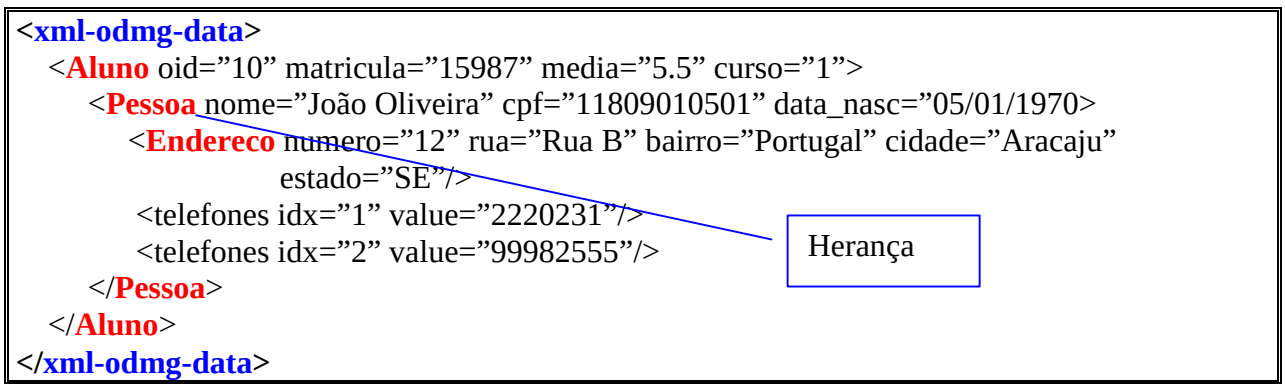

Figura 4.23: Representação de instâncias de classes que possuem herança, em XML-ODMG

A classe `Aluno` herda características da classe `Pessoa`, desta forma o elemento `Pessoa` representará a classe `Pessoa` e os valores de seus atributos. A representação da

herança consiste em representar a classe herdada dentro da instância da classe que herda conforme as definições da linguagem XML-ODMG.

Representando valores nulos

Os valores nulos são representados através da *string* “#null” no lugar do valor do atributo, como mostra o exemplo da figura 4.24. A representação de um valor nulo só pode ser realizada caso a definição deste mesmo atributo no XML-ODMG de Esquema o defina como não obrigatório.

```
<xml-odmg-data>
  <Aluno oid=”10” matricula=”15987” media=”5.5” curso=”1”>
    <Pessoa nome=”João Oliveira” cpf=”11809010501” data_nasc=”05/01/1970>
      <Endereco numero=”12” rua=”Rua B” bairro=”#null” cidade=”Aracaju”
                estado=”SE”/>
    </Pessoa>
  </Aluno>
</xml-odmg-data>
```

Figura 4.24: Representação de valores nulos em XML-ODMG

4.2.3 A Linguagem XML-ODMG de Transformação e Resolução de Conflitos

O documento produzido por esta linguagem tem por finalidade descrever o esquema do banco de dados de destino, como também as regras para a resolução de conflitos que possam existir entre os esquemas de origem e destino. Mais precisamente, este documento define o mapeamento entre as informações contidas no XML-ODMG de Dados para os elementos persistentes no banco de dados de destino. A mesma definição para representação de classes, atributos, relacionamentos, *interfaces* e heranças utilizadas no XML-ODMG de Esquema é seguido por este documento, com a diferença que novos elementos são adicionados a fim de definir regras para a resolução de conflitos. A figura 4.5 mostra a hierarquia de elementos de um documento XML-ODMG de Transformação e Resolução de Conflitos. Todos os elementos serão explicados mais adiante neste capítulo.

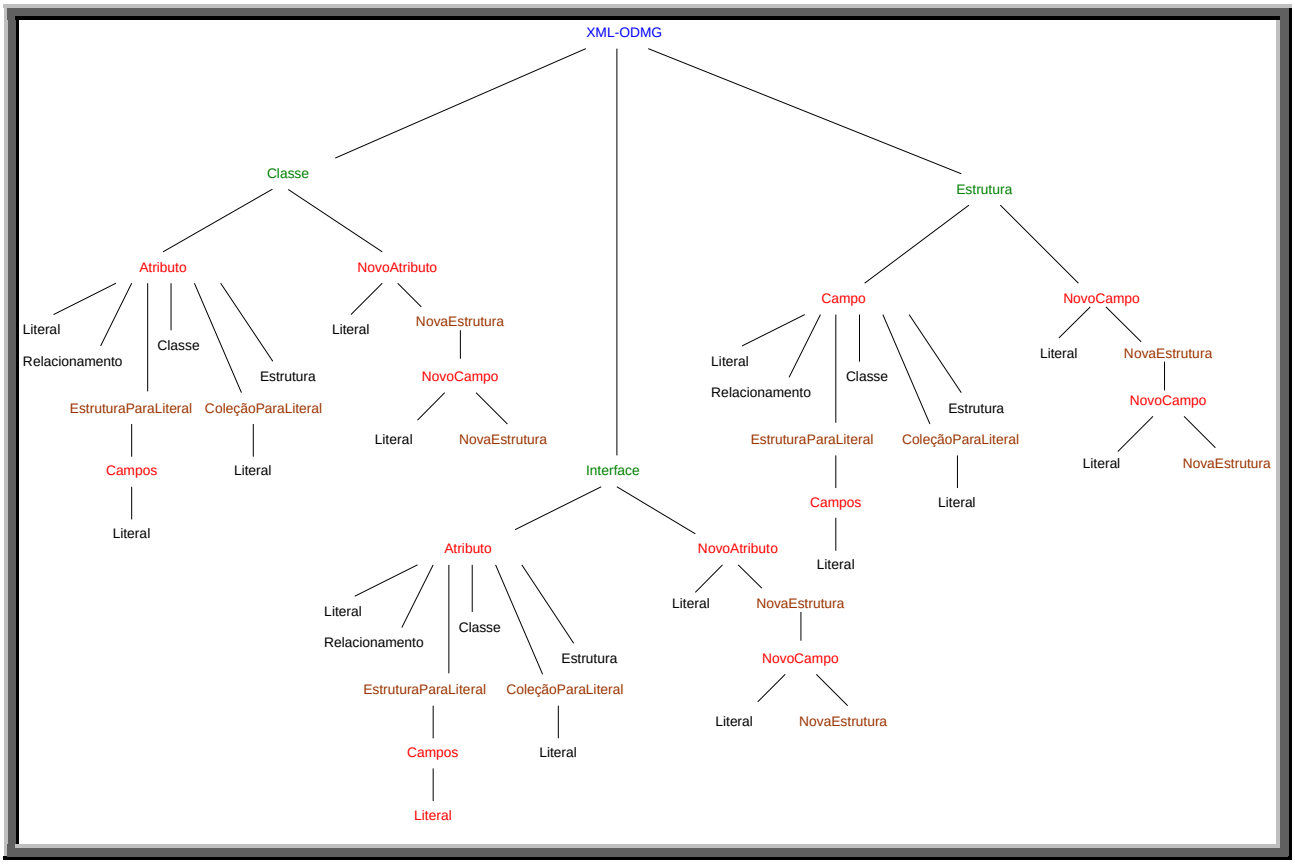

Figura 4.25: Hierarquia dos elementos do XML-ODMG de Transformação e Resolução de Conflitos

A importância da definição de regras para a resolução de conflitos oferece a flexibilidade necessária para que o SIBD não inviabilize a operação de intercâmbio de dados na ocorrência de conflitos.

Em um modelo orientado a objeto estes conflitos se manifestam a nível de atributo, a nível de entidades (entende-se por entidades classes e tipos estruturados) e de atributos *vs* entidades [KIM 93]. Nesta seção serão mostrados os conflitos que podem ser resolvidos através da linguagem XML-ODMG e como é realizado este procedimento.

As figuras 4.26 e 4.27 mostram a definição do esquema de destino destacando os pontos de conflitos existentes com o esquema de origem da seção 4.3. O DTD utilizado para a construção de um documento XML-ODMG de Transformação e Resolução de Conflitos está no apêndice A e o documento completo deste modelo no apêndice B.

struct Desconto {
  char tipo_desconto;
  float val_desconto;
  char situacao;
	struct Carga_hor_turno {
	  attribute char turno**;**
	  attribute short carga_hor**;**
	}

```
}

class Pessoa {
  attribute string nome;
  attribute char cpf;
  attribute date data_nasc;
  attribute string endereco ;
  attribute set<string> telefones;
}

class Estudante  extends Pessoa
   ( extent estudantes
     key codigo ) {
    attribute short codigo;
    attribute float media_geral;
    attribute string repreensoes;
    attribute Desconto desconto;
    attribute set<
       struct Matricula {
            float nota,
            short faltas,
            relationship set<Disciplina> faz
               inverse Disciplina::é_feita_por } >
        matriculas;
   relationship Curso cursa
       inverse Curso::é_cursado_por;
    void matricular( in short cod_disc );
    void cancelar_matricula( in short cod_disc );
    void transferir_matricula (
                    in short cod_disc_velho,
                    in short cod_disc_novo );

}

Curso Curso
  ( extent cursos
     key cod_curso )  {
  attribute short cod_curso;
  attribute string nom_curso;
  attribute char turno;
  attribute date data_reconhecimento;
  relationship set<Estudante> é_cursado_por
     inverse Estudante::cursa;
}
```

```
struct Titulacao {
   attribute string titulo;
   attribute string instituicao;
   attribute date inicio;
   attribute date fim;
}

class Professor extends Pessoa
   ( extent professores
     key cod_prof ) {
   attribute short cod_prof;
   attribute Carga_hor_turno
                carga_hor_diponivel;
   attribute string centro;
   attribute set<Titulacao> titulacoes;
   relationship set<Disciplina> ensina
      inverse é_ensinada_por :: Disciplina
}


class Disciplina
   ( extent disciplinas,
     key cod_disc  ) {
   attribute short cod_disc;
   attribute string nom_disc;
   attriburte short carga_horaria;
   relationship set<Estudante> é_feita_por
      inverse Estudante::faz;
   relationship set<Professor> é_ensinada_por
      inverse Professor::ensina;
}
```

Figura 4.26: Definição ODL analítico do esquema de destino

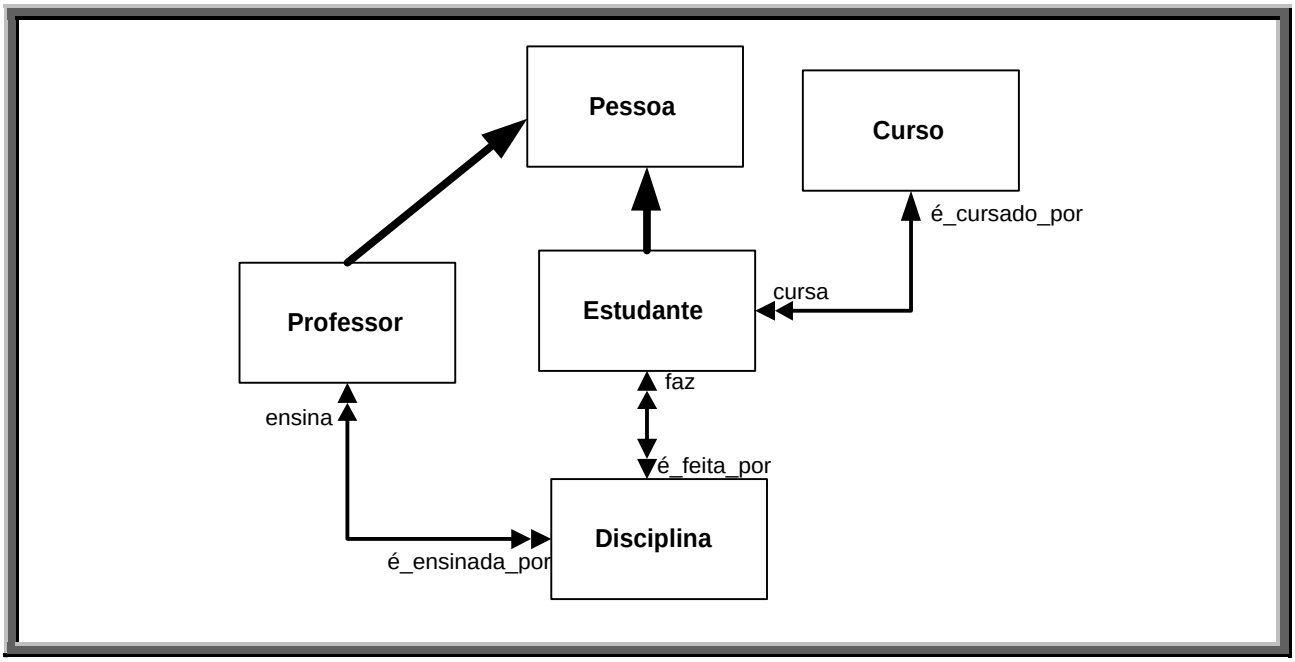

Figura 4.27: Diagrama ODL do esquema de destino

Conflitos de entidades

Os conflitos de entidades suportados pela XML-ODMG são:

- nomes de entidades diferentes que representam a mesma informação;
- mesmo nome de entidades que representam informações diferentes;
- número diferente de atributos entre entidades que representam a mesma informação;

Nomes de entidades diferentes que representam a mesma informação

Este conflito se manifesta quando duas classes que representam a mesma informação (classes equivalentes) estão definidas em seus respectivos esquemas de banco de dados com nomes diferentes. Segundo [KIM 93] a resolução para este conflito é renomear uma das classes, de forma a ter as duas com o mesmo nome. A XML-ODMG define um novo atributo no elemento que representa a classe conflitante, chamado de `newName`, que deve conter o nome da classe no esquema de destino. Caso não exista este conflito a utilização deste atributo não é obrigatória.

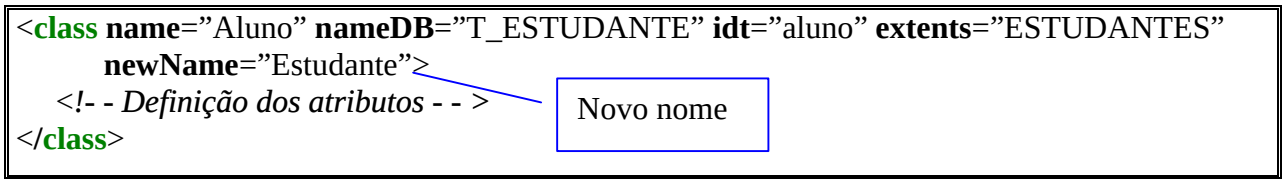

Figura 4.28: Resolução do conflito para nome de entidades diferentes que representam a mesma informação, em XML-ODMG

O exemplo da figura 4.28 define que toda instância da classe `Aluno` presente no XML-ODMG de Dados deve ser transformada em instâncias da classe `Estudante` definido pelo atributo `newName`. Os atributos `nameDB` e `extents` fazem referência ao nome da classe `Estudante` no banco de dados de destino, já o atributo `name` à classe do banco de dados de origem.

Mesmo nome de entidades que representam informações diferentes

Este conflito se manifesta quando duas classes (uma no banco de dados de origem e outra no banco de dados de destino) de mesmo nome representam informações diferentes. A resolução para este problema é renomear a classe do modelo de origem para o nome da classe do modelo de destino que ela realmente seja equivalente [KIM 93]. Por exemplo, no modelo de origem a classe `Funcionario`, que representa os professores neste modelo, é equivalente à classe `Professor` no modelo de destino, porém neste modelo a classe `Funcionario` representa os funcionários que não são professores, desta forma configura-se um conflito de nomes uma vez que os professores no modelo de origem são representados pela classe `Funcionario` enquanto que no modelo destino os professores são representados pela classe `Professor`, e não pela classe `Funcionario`. Para resolver este conflito em XML-ODMG utiliza-se o atributo `newName`, para renomear a classe `Funcionario` do modelo de origem para `Professor`, fazendo a relação com a classe `Professor` do modelo de destino.

Quantidade diferente de atributos entre entidades que representam a mesma informação

Este tipo de conflito aparece quando duas classes equivalentes, em banco de dados diferentes, não possuem o mesma quantidade de atributos. A diferença na quantidade de

atributos pode ocorrer a mais ou menos em relação a uma das classes. A linguagem XML-ODMG assume sempre a classe do esquema de destino como referência para verificar a ausência ou a sobra de atributos em relação a sua classe equivalente.

Quando este tipo de conflito se manifesta na ausência de atributos, a sua resolução, segundo [KIM 93], é definir valores que serão utilizados para preencher os atributos que não existem na classe do esquema de destino. Para resolver este conflito, faz-se uso das marcações `newAttribute`, quando for um atributo do tipo literal ou `newStruct` quando o atributo for de um de um tipo estruturado, em conjunto com a marcação `newValue`.

Inicialmente será exemplificado a ausência de atributo do tipo literal, posteriormente será feita uma demonstração da resolução deste conflito para o atributo do tipo estruturado.

As tabelas 4.9 e 4.10 apresentam os atributos que devem ser utilizados nas marcações `newAttribute` e `newValue` respectivamente, que devem ser utilizadas para a resolução deste conflito para atributos do tipo literal.

Atributo	**Opcional**	**Valor *Default***	**Valores**	**Descrição**
name	Não			Nome do atributo.
type	Não	Literal	literal struct	Tipo de atributo. Pode ser uma literal (valor “literal”) ou um tipo definido pelo usuário (valor “struct”).
nameDB	Não			Nome do atributo no banco de dados.

Tabela 4.9: Definição dos atributos do elemento `newAttribute`

Atributo	**Opcional**	**Valor *Default***	**Valores**	**Descrição**
val	Não			Define o valor do atributo que vai ser incluído no banco de dados para todas instâncias da classe representada.

Tabela 4.10: Definição do atributo do elemento `newValue`

```
<class name=”Curso” idt=”curso” nameDB=”T_CURSO” extents=”CURSO”>
  <!- - Definição dos atributos comuns entre as classes (origem e destino) - - >
  <newAttribute name=”turno” type=”literal” nameDB=”TURNO”>
     <literal type=”char”/>
     <newValue val=”M”/>
  </newAttribute>
</class>
```

Valor para preenchimento do atributo

Figura 4.29: Resolução, em XML-ODMG, do conflito de quantidade diferente de atributos entre classes que representam a mesma informação

No exemplo da figura 4.29 a classe `Curso` do esquema de destino possui um atributo, `turno`, o qual não existe na classe equivalente no esquema de origem. Através do elemento `newAttribute` é possível definir o atributo `turno` do esquema de destino e com a marcação `newValue` o valor que deve preencher este atributo no banco de dados, visto que não existem valores oriundos do banco de dados de origem. Desta forma, todas as instâncias da classe `Curso` incluídas no banco de dados de destino conterão o valor “M” para o atributo `turno`. Uma classe pode conter nenhuma ou várias marcações `newAttribute`, o que determina este número é a quantidade de atributos ausentes.

A ausência de atributos pode se manifestar também, para atributos de um tipo estruturado. Para resolver este conflito, a linguagem XML-ODMG utiliza o elemento `newAttribute`, com a diferença que o seu atributo `type` deve possuir o valor “struct” ao invés do valor “literal”. O conteúdo da marcação `newAttribute`, nesta situação, será formado pela marcação `newStruct`, responsável por definir o tipo estruturado. Para representar os campos que formam este tipo estruturado utiliza-se a marcação `newField`, responsável por indicar o nome, o tipo e valor do dado que será incluído no banco de dados.

As tabelas 4.11 e 4.12 apresentam os atributos que devem ser utilizados nas marcações `newStruct` e `newField`.

Atributo	**Opcional**	**Valor *Default***	**Valores**	**Descrição**
name	Não			Nome conceitual do tipo estruturado
nameDB	Não			Nome do tipo estruturado no banco de dados

Tabela 4.11: Definição dos atributos do elemento `newStruct`

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Nome do campo do tipo estruturado
nameDB	Não			Nome do campo do tipo estruturado no banco de dados
type	Não	literal	literal struct	Tipo do campo no tipo estruturado

Tabela 4.12: Definição dos atributos do elemento `newField`

```
<class name="Professor" idt="professor" nameDB="T_PROFESSOR"
       extent="PROFESSORES">
  <! - - Definições dos atributos comuns - ->
  <newAttribute name="carga_hor_turno" nameDB="CARGA_HOR_TURNO"
                type="struct">
     <newSruct name="Carga_hor_turno" nameDB="T_CARGA_HOR_TURNO">
        <newField name="turno" nameDB="TURNO" type="literal">
           <literal type="char"/>
           <newValue val="M"/>
        </newField>
        <newField name="carga_hor" nameDB="CARGA_HOR" type="literal">
           <literal type="short"/>
           <newValue val="20"/>
        </newField>
     </newStruct>
  </newAttribute>
</class>
```

Novo tipo estruturado

Campo do novo tipo estruturado

Valor para preenchimento do campo

Figura 4.30: Resolução, em XML-ODMG, do conflinto de ausência de atributos quando se manifesta para atributos formados por um tipo estruturado

Neste exemplo da figura 4.30, o esquema de origem não possui o atributo `carga_hor_turno` na classe `Professor`, assim devem ser definidos valores para preencher este atributo no esquema de destino. Estes valores são definidos através dos elementos `newStruct`, que representa o tipo estruturado `Carga_hor_turno`, e `newValue` que determina os valores que serão incluídos no banco de dados para cada campo desta entidade.

Ao tempo que a ausência de atributo entre classes equivalentes pode ocorrer, este conflito também se manifesta entre tipos estruturados equivalentes em bancos de dados diferentes. A mesma solução utilizada para resolver este conflito em relação às classes também se aplica aos tipos estruturados. Em XML-ODMG este conflito é resolvido

utilizando a marcação `newField` para representar o campo ausente e definir o valor que será incluído no banco de dados de destino.

```
<struct name=”Desconto” nameDB=”T_DESCONTO” idt=”desconto”>
    <!- - Definição dos atributos comuns - - >
    <newField name=”suspenso” nameDB=”SUSPENSO” type=”literal”>
        <literal type=”char”/>
        <newValue val=”A”/>
    </newField>
</struct>
```

Figura 4.31: Resolução, em XML-ODMG, do conflito de ausência de campos para tipos estruturados

Neste exemplo da figura 4.31, o tipo estruturado `Desconto`, do esquema de destino, possui o campo `suspenso` que não existe no esquema de origem. Assim este campo é representado através da marcação `newField` que, em conjunto com a marcação `newValue`, determinará o valor que deve ser atribuido a este campo, quando o tipo estruturado `Desconto` for referenciado.

Sempre que ocorrer um conflito de ausência de atributo ou campo, deve-se obrigatoriametne fazer o seu tratamento no XML-ODMG de Transformação ou Resolução de Conflito, mesmo que este atributo seja opcional. Caso não seja necessário atribuir um determinado valor ao atributo ou ao campo, este deve ser definido como nulo utilizando o indicador “#null”, conforme é apresentado no exemplo da figura 4.32.

```
<class name=”Curso” idt=”curso” nameDB=”T_CURSO” extents=”CURSO”>
  <!- - Definição dos atributos comuns entre as classes (origem e destino) - - >
  <newAttribute name=”turno” type=”literal” nameDB=”TURNO”>
      <literal type=”char”/>
      <newValue val=”#null”/>
  </newAttribute>
</class>
```

Representação do valor nulo

Figura 4.32: Representação, em XML-ODMG, para um valor nulo na resolução do conflito de quantidade diferente de atributo

Conflitos de Atributos

Conflitos de atributos são diferenças encontradas entre atributos de classes equivalentes que representam a mesma informação ou entre campos de tipos estruturados. Os conflitos de atributos suportados pela XML-ODMG são:

- nomes diferentes para atributos equivalentes;
- mesmo nome para atributos não equivalentes;
- tipos de literais diferentes para atributos equivalentes;
- estruturas diferentes para atributos equivalentes.

Nomes diferentes para atributos equivalentes

Este conflito se manifesta quando atributos que representam a mesma informação (atributos equivalentes) em classes equivalentes não possuem o mesmo nome. A solução deste conflito, proposta por [KIM 93], é renomear um destes atributos para o mesmo nome do seu equivalente. A XML-ODMG define o atributo `newName`, na marcação que representa o atributo de uma classe ou o campo de um tipo estruturado, que deve conter o nome do atributo ou campo do esquema de destino equivalente. A resolução deste conflito é ilustrada no exemplo da figura 4.33.

```
<class name="Aluno" newName="Estudante" idt="aluno" nameDB="T_ESTUDANTE"
       extents="ESTUDANTES">
   <attribute name="matricula" newName="codigo" nameDB="CODIGO" type="literal">
       <literal type="short"/>
    </attribute>
    <!- - Definição de outros atributos - - >
</class>
```

Indica o nome no esquema de destino

Figura 4.33: Resolução, em XML-ODMG, do conflito para atributos de nomes diferentes que representam a mesma informação

As classes `Aluno`, do esquema de origem, e `Estudante`, do esquema de destino, são equivalentes, e seus atributos, `matricula` e `codigo`, representam a mesma informação. Desta forma configura-se um conflito de nomes a nível de atributo e é resolvido com a adição do atributo `newName` no elemento `attribute`, que deve possuir o nome do atributo do esquema de destino, neste exemplo "codigo". Através do valor do atributo `name`

e `newName` é possível fazer a relação entre os atributos equivalentes nos esquemas de origem e de destino.

A mesma solução deve ser aplicada quando existe um conflito de nomes entre campos de tipos estruturados equivalentes.

```
<struct name=”Desconto” nameDB=”T_DESCONTO” idt=”desconto”>
   <field name=”valor” newName=”val_desconto” nameDB=”VAL_DESCONTO”
        type=”literal”>
        <literal type=”float”/>
    </field>
    <!- - Definição dos demais campos - ->
</struct>
```

Figura 4.34: Resolução, em XML-ODMG, para o conflito de campos de tipos estruturados que possuem nomes diferentes que representam a mesma informação

Neste exemplo da figura 4.34 o campo `val_desconto` do tipo estruturado `Desconto` no esquema de destino é equivalente ao campo `valor` do tipo estruturado `Desconto` no esquema de origem. Desta forma, o atributo `newName` deve possuir o valor “val_desconto” a fim da fazer a relação com o campo equivalente no esquema de origem “valor”.

Mesmo nome para atributos não equivalentes

Este conflito aparece quando atributos ou campos de mesmo nome em entidades equivalentes não representam a mesma informação. A resolução para este conflito é renomear um dos atributos ou campos para um valor que represente o seu equivalente utilizando a marcação `newName`.

Tipos de literais diferentes para atributos equivalentes

Este conflito se apresenta quando atributos ou campos do tipo literal que representam a mesma informação, estão definidos em domínios diferentes. Por exemplo, um atributo pode estar definido como *string* enquanto que o seu equivalente em outro esquema está definido como *short.* A solução para este conflito é realizar a transformação de tipos, caso seja possível, de forma a adequar os valores equivalentes[KIM 93].

A tabela 4.13 ilustra as transformações suportadas pela XML-ODMG. Onde o valor "S" indica que a transformação pode ser realizada irrestritamente, "A" que pode ser realizada a depender do valor de origem e "N" que não pode ser realizada.

Para	long	short	Double	Float	string	boolean	char	date	timestamp
De									
Long	S	A	S	S	S	N	S	N	N
Short	S	S	S	S	S	N	S	N	N
Doublé	N	N	S	A	S	N	S	N	N
Float	N	N	S	S	S	N	S	N	N
String	A	A	A	A	S	N	A	A	A
Boolean	N	N	N	N	N	S	N	N	N
Char	A	A	A	A	S	N	S	N	N
Date	N	N	N	N	A	N	N	S	S
timestamp	N	N	N	N	A	N	N	N	S

Tabela 4.13: Tabela de transformações de tipos da XML-ODMG

A ocorrência do valor "A" na tabela 4.13 se justifica pois, existem algumas transformações que só serão realizadas a depender do valor. Por exemplo, o valor "CASA" do tipo *string* não pode ser convertido para o tipo *float*, porém o valor "12.5" também do tipo *string* pode ser transformado em um valor do tipo *float* sem restrições.

Estruturas diferentes para atributos equivalentes

Este conflito se apresenta quando atributos ou campos equivalentes são formados por estruturas distintas. Estas diferenças de estruturas ocorrem quando um atributo é uma literal e o seu equivalente é uma coleção, ou vice-versa, ou quando um atributo é uma literal e o seu equivalente é um tipo estruturado ou vice-versa [KIM 93]. A XML-ODMG resolve conflitos no sentido de uma coleção para literal, ou vice-versa, ou de um tipo estruturado para literal, neste último a ordem inversa não é suportada.

Coleção para literal ou literal para coleção

Acontece quando um atributo ou um campo é uma coleção no esquema de origem e o seu equivalente no esquema de destino é um literal ou vice-versa.

Duas situações deste conflito não são suportadas pela linguagem XML-ODMG de Transformação e Resolução de Conflitos:

1. se a coleção não for um conjunto de literais, podendo ser um conjunto de tipos estruturados ou de relacionamentos;
2. se a coleção for formada por um conjunto de literais de um tipo diferente do tipo do seu equivalente e a transformação não for possível segundo a tabela 4.13. Por exemplo, se no esquema de origem um atributo é formado por um conjunto de valores *string* e o seu equivalente no esquema de destino for do tipo *boolean*.

Coleção para literal

A solução para este conflito, consiste em escolher um item dentro da coleção do esquema de origem que vai ser utilizado para preencher o atributo da classe ou campo de um tipo estruturado no banco de dados de destino. A XML-ODMG define o elemento `collectionToLiteral` para resolver este conflito. O elemento que representa o atributo ou campo onde ocorre este conflito deve ter o valor do seu atributo `type` igual a "collectionToLiteral".

```
<class name="Curso" idt="curso" nameDB="T_CURSO" extents="CURSO">
  <!- - Definição dos atributos comuns entre as classes (origem e destino) - - >
  <attribute name="datas_reconhecimento" newName="data_reconhecimento"
             nameDB="DATA_RECONHECIMENTO" type="collectionToLiteral"/>
     <collectionToLiteral idx="1">
        <literal type="date"/>
     </collectionToLiteral>
  </attribute>
</class>
```

Índice que indica qual elemento da coleção vai ser utilizado

Figura 4.35: Resolução, em XML-ODMG, do conflito *coleção para literal*

O elemento `collectionToLiteral` indica através do atributo `idx` um valor dentro da coleção que será utilizado para preencher o valor do atributo equivalente e o tipo

deste através do elemento `literal`. Segundo o exemplo da figura 4.35 será utilizado o valor do item de índice 1 presente no documento XML-ODMG de Dados.

Literal para coleção

Para resolver este conflito o valor do atributo ou campo do esquema de origem é incluído como um novo item da coleção do atributo ou campo no esquema de destino. Para isto o elemento que representa o atributo da classe ou o campo de um tipo estruturado deve conter o valor "literalToCollection" no atributo `type`.

```
<class name="Aluno" newName="Estudante" nameDB="T_ESTUDANTE"
       extents="ESTUDANTES">
   <!- - Definições dos atributos da classe - - >
    <attribute name="repreensão" newName="repreensoes" nameDB="REPREENSOES"
               type="literalToCollection" collection="yes" collectionType="set">
        <literal type="string"/>
    </attribute>
</class>
```

Transformação de literal para coleção

Figura 4.36: Resolução, em XML-ODMG, para o conflito *literal para coleção*

No exemplo da figura 4.36 o valor do atributo `repreensao`, no esquema de origem, é incluído como um item da coleção presente no atributo `repreensoes`.

Tipo estruturado para literal

Este conflito se manifesta quando um atributo, no esquema de origem, é de um tipo estruturado e o seu equivalente, no esquema de destino, é uma literal. A linguagem XML-ODMG não suporta o conflito no sentido inverso. As soluções deste conflito, propostas por [KIM 93], é escolher um dos campos do tipo estruturado que melhor represente o atributo do esquema de destino ou concatenar alguns (ou todos) campos do tipo estruturado, a fim de se obter um valor único.

A linguagem XML-ODMG suporta estas duas soluções através da utilização da marcação `structToLiteral` em conjunto com a marcação `fields`. As tabelas 4.14 e 4.15 apresentam os atributos destas marcações respectivamente.

Atributo	Opcional	Valor *Default*	Valores	Descrição
structName	Não			Nome do tipo estruturado no esquema de origem

Tabela 4.14: Definição dos atributos do element `structToLiteral`

Atributo	Opcional	Valor *Default*	Valores	Descrição
name	Não			Campo dentro do tipo estruturado
order	Não			Indica a posição do campo na concatenação
caracter	Não		<espaço em branco>	Indica o *caracter* que será utilizado para separar os campos no momento da concatenação

Tabela 4.15: Definição dos atributos do element `fields`

```
<class name=”Pessoa” nameDB=”T_PESSOA” idt=”pessoa”>
   <!- - Definição dos atributos - - >
    <attribute name=”endereco” nameDB=”T_ENDERECO”  type=”structToLiteral”>
        <literal type=”string”/>
        <structToLiteral  structName=”Endereco”>
           <fields name=”rua” order=”1” caracter=” “>
               <literal type=”string”/>
           </fields>
           <fields name=”numero” order=”2” caracter=” “>
               <literal type=”string”/>
           </fields>
        </structToLiteral>
    </attribute>
</class>
```

Tipo do atributo

Campos que irão se juntar para formar um valor único

Campos que irão se juntar para formar um valor único

Figura 4.37: Resolução, em XML-ODMG, do conflito *tipo estruturado para coleção*

O atributo `endereco` da classe `Pessoa` no esquema de destino é formado por uma literal do tipo *string*, enquanto que seu equivalente no esquema de origem é um tipo estruturado. Utilizando-se a marcação `structToLiteral` serão informados os campos do tipo estruturado que serão concatenados para formar um valor único. Os campos que serão utilizados são determinados através dos elementos `fields` que ocorrem dentro da marcação `structToLitertal`. No exemplo da figura 4.37 o valor do campo `rua` e o do campo

`numero` serão concatenados, nesta ordem, formando um valor único que será incluído ao atributo `endereco` no esquema de destino.

Conflito de entidades *vs* atributos

Este conflito se manifesta quando um atributo de uma classe ou um campo de um tipo estruturado, do tipo literal, tem como equivalente em outro esquema um relacionamento para outra classe. O desafio da resolução deste conflito é transformar o relacionamento com uma classe em um atributo do tipo literal. Segundo [KIM 93] a resolução para esta diferença se assemelha a anterior, deve-se escolher um atributo da classe que está sendo referenciada (através do relacionamento) cujo valor vai preencher o atributo equivalente ou um conjunto de atributos desta classe que serão concatenados.

Para resolver este conflito, em XML-ODMG, deve-se utilizar o elemento `structToLiteral` e definir o valor do atributo `type` da marcação que representa o atributo ou campo para o valor "relationShipToLiteral", como mostra o exemplo da figura 4.38.

```
<class name="Professor" nameDB="T_PROFESSOR" idt="professor"
       extents="PROFESORES">
   <!- - Definição dos atributos - ->
   <attribute name="centro" nameDB="CENTRO" type="relationShipToLiteral">
      <literal type="string"/>
       <structToLiteral structName="Centro">
           <fields name="nom_centro" order="1" caracter="">
               <literal type="string"/>
           </fields>
       </structToLiteral>
   </attribute>
</class>
```

Nome da classe de relacionamento

Figura 4.38: Resolução, em XML-ODMG, do conflito *entidade vs atributos*

O atributo `centro` da classe `Professor` do esquema de destino é do tipo *string*, enquanto que o seu equivalente, no esquema de origem, é a classe `Centro`, referenciada pela classe `Professor`. Assim é escolhido um atributo da classe `Centro`, no exemplo da figura 4.38, `nom_centro`, cujo valor vai preencher o atributo `centro` no esquema de destino.

4.6 A Linguagem XML-ODMG no intercâmbio de dados

Utilizar a linguagem XML-ODMG nas operações de intercâmbio de dados entre banco de dados consiste basicamente em cinco ações:

1. Definição do XML-ODMG de Esquema que representa a base de dados do banco de dados de origem;
2. Geração do XML-ODMG de Dados a partir de uma base de dados conforme as definições do XML-ODMG de Esquema, através da ação de um SIBD;
3. Transmissão do XML-ODMG de Dados do ambiente de origem para o ambiente de destino. Nesta ação o meio de transmissão não é levado em consideração;
4. Definição do XML-ODMG de Transformação e Resolução de Conflito, o qual vai representar o mapeamento do XML-ODMG de Dados para o esquema do banco de dados de destino;
5. Transferência das informações do XML-ODMG de Dados para uma base de dados observando as definições do XML-ODMG de Transformação e Resolução de Conflitos, através da ação de um SIBD.

O esquema gráfico destas ações pode ser observada na figura 4.39.

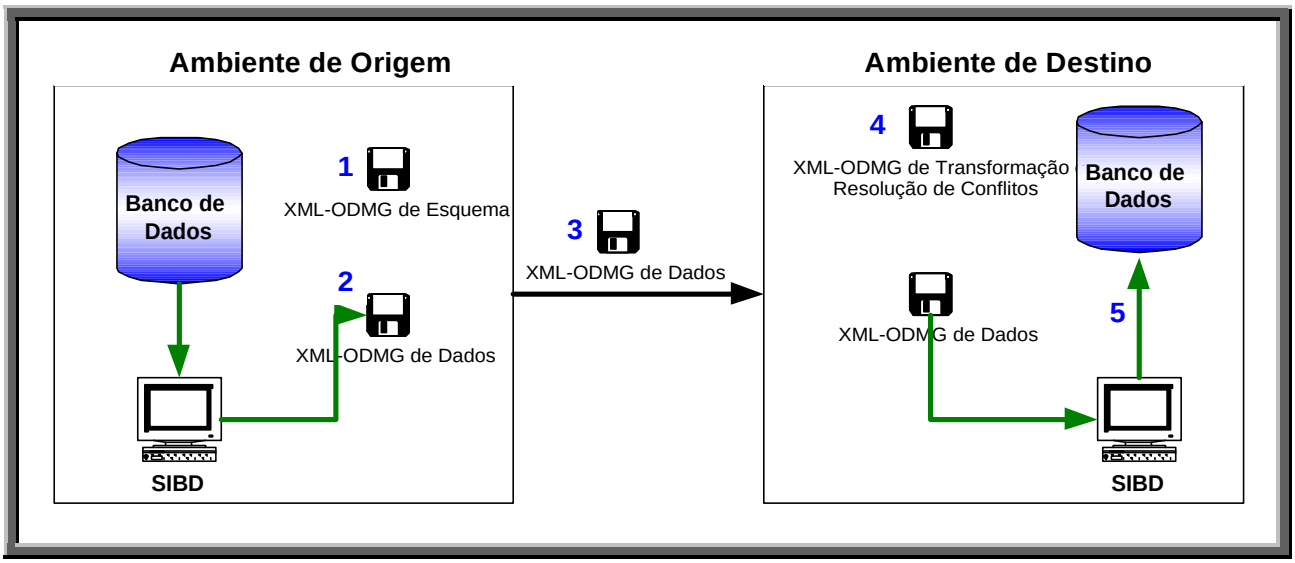

Figura 4.39: Utilização da linguagem XML-ODMG no intercâmbio de dados

4.7 Comparação com linguagens existentes

A linguagem OIFXML, apresentado na seção 3.4.1, possui os mesmos propósitos da XML-ODMG. A principal diferença entre ambas é que XML-ODMG gera dois documentos intermediários, um para o esquema (XML-ODMG de Esquema) e o outro para os dados (XML-ODMG de Dados), enquanto que OIFXML trabalha com um único documento intermediário, descrevendo ao mesmo tempo esquema e dados. Utilizar documentos distintos para representar esquema, incluindo-se os meta-dados, e dados possibilita o tratamento das informações com um maior nível de detalhamento, gerando também documentos mais legíveis uma vez que os dados propriamente ditos não se misturam com as informações que os definem [WIL 00].

As figuras 4.40 e 4.41 mostram a definição da classe curso (definição da figura 4.1) em OIFXML e XML-ODMG respectivamente.

OIFXML

```
<oif_file>
  <odmg_object oid="Curso1">
    <class>Curso</class>
      <contents>
        <attribute name="cod_curso">
           <value>
              <short val="23"/>
           </value>
        </attribute>
        <attribute name="nom_curso">
           <value>
             <string val="Informática"/>
          </value>
        </attribute>
        <attribute name="carga_horaria">
          <value>
            <short val="40"/>
          </value>
        </attribute>
        <!-- O tipo Date não é suportado pelo OIFXML -->

        <relationship name="e_cursado_por">
          <links to="Aluno1 Aluno2 Aluno3" type="set"/>
        </relationship>
    </contents>
  </odmg_object>
</oif_file>
```

Figura 4.40: Definição da classe curso em OIFXML

XML-ODMG de Esquema

```
<xml-odmg>
  <class name="Curso" idt="curso" nameDB="T_CURSO" extent="CURSOS">
    <attribute name="cod_curso" nameDB="COD_CURSO" key="yes" type="literal"
               length="4">
      <literal type="short"/>
    </attribute>
    <attribute name="nom_curso" nameDB="NOM_CURSO" type="literal"
               length="80">
      <literal type="string"/>
    </attribute>
    <attribute name="carga_horaria" nameDB="CARGA_HORARIA" type="literal"
               length="4">
      <literal type="short"/>
    </attribute>
    <attribute name="datas_reconhecimento" nameDB="DATAS_RECONHECIMENTO"
               type="literal" collection="yes" collectionType="set">
      <literal type="date"/>
    </attribute>
    <attribute name="e_cursado_por" nameDB="E_CURSADO_POR"
               type="relationship" collection="yes" collectionType="set" >
      <relationship relationshipRef="cursa" name="e_cursado_por"
                    classRef="aluno"/>
    </attribute>
  </class>
</xml-odmg>
```

XML-ODMG de Dados

```
<xml-odmg-data>
  <curso oid="1" cod_curso="1" nom_curso="Informática" carga_horaria="40">
    <datas_reconhecimento idx="1" value="05/10/2002"/>
    <datas_reconhecimento idx="2" value="05/11/2002"/>
    <e_curada_por idx="1" value="245"/>
    <e_curada_por idx="2" value="234"/>
    <e_curada_por idx="3" value="123"/>
  </curso>
</xml-odmg-data>
```

Figura 4.41: Definição da classe curso em XML-ODM**G**

Note-se que a linguagem XML-ODMG oferece um maior poder de representação do esquema de dados em relação a OIFXML, possibilitando uma melhor identificação e recuperação dos dados. Além disso, verifica-se que o documento de dados gerado na linguagem XML-ODMG (XML-ODMG de Dados) se apresenta de forma mais simples e com

menor dimensão em relação ao documento gerado na OIFXML, favorecendo a perfomance na recuperação dos dados.

A tabela 4.16 apresenta uma comparação entre XML-ODMG e OIFXML.

Característica	XML-ODMG	OIFXML
Estrutura dos documentos	Os dados e o esquema são tratados em documentos diferentes.	Os dados e o esquema são representados em um único documento.
Tamanho do documento de dados	Os documentos de dados (XML-ODMG de Dados) são menores, uma vez que apenas os dados, propriamente ditos, são representados.	São documentos maiores uma vez que os dados são representados juntamente com as informações do esquema.
Legibilidade	Por definir um documento para representar os dados e outro para representar o esquema, obtém-se documentos mais legíveis.	Organizando dados e esquema em um único documento, obtém-se documentos menos legíveis.
Resolução de Conflitos	Define regras para resolver possíveis conflitos existentes entre os esquemas de origem e destino.	Não define regras para resolução de conflitos, podendo inviabilizar o processo de intercâmbio de dados.

Tabela 4.16: Diferenças entre XML-ODMG e OIFXML

4.8 Conclusão

Diante do exposto neste capítulo, pode-se concluir que a linguagem XML-ODMG é capaz de descrever um esquema ODL/ODMG e os dados correspondentes por meio de documentos XML, a fim de utilizá-los em operações de intercâmbio de dados entre banco de dados. XML-ODMG utiliza-se de documentos diferentes para representar o esquema e os dados, proporcionando documentos com uma maior riqueza de representação, mais legíveis e menores em relação a OIFXML. Além de representar o esquema e os dados, a XML-ODMG também possibilita a definição de regras para a resolução de conflitos existentes entre os banco de dados de origem e destino, oferecendo flexibilidade no processo de intercâmbio de dados.

A riqueza no detalhamento das informações existentes no banco de dados e a possibilidade de tratar possíveis conflitos, colocam a linguagem XML-ODMG como a melhor solução para representar informações de um modelo de dados orientado a objeto em comparação a linguagens criadas para a mesma finalidade como a OIFXML.

Capítulo 5

XML-ODBMS

5.1 Introdução

XML-ODBMS é Sistema de Intercâmbio de dados entre Banco de Dados utilizando XML. O que o difere de outros SIBDs é a característica de suporte a banco de dados orientados a objetos e objeto-relacionais, por utilizar o padrão ODMG para representar os dados e permitir a definição de regras para a resolução de conflitos entre os esquemas de banco de dados envolvidos no processo de intercâmbio de dados. O fato de oferecer suporte a BD OO e OR, não exclui os BD relacionais, uma vez que o modelo objeto-relacional é uma extensão do relacional.

Neste capítulo descreveremos XML-ODBMS apresentando sua arquitetura, seus requisitos funcionais e não funcionais, e detalhes do seu desenvolvimento.

5.2 Objetivos

O principal objetivo do XML-ODBMS é disponibilizar, para desenvolvedores de aplicações necessitando fazer intercâmbio de dados entre bancos de dados orientados a objetos e bancos de dados objetos-relacionais, um conjunto de APIs que possam ser incluídas no código fonte de suas aplicações a fim de potencializá-las com funções de intercâmbio de dados [GRA 02].

As APIs XML-OBMS abstraem toda a complexidade da construção de documentos intermediários para a importação/exportação de dados entre SGBDs, e todas as outras operações de transações de intercâmbio de dados.

5.3 Requisitos Funcionais

Os requisitos funcionais são as principais funções que devem ser desempenhadas pelo XML-ODBMS a fim de atender aos objetivos propostos na seção 5.2. :

- **Representação de esquema de banco de dados em XML**

O XML-ODBMS deve prover uma boa representação de esquema de banco de dados sob a forma de documentos XML. Para isso, XML-ODBMS utiliza-se da linguagem XML-ODMG apresentada no capítulo 4.

- **Representação de dados em XML**

O XML-ODBMS também deve oferecer uma boa representação dos dados, extraídos de uma base de dados, em XML. A linguagem XML-ODMG dá também suporte à representação de dados.

- **Definição de regras para a resolução de conflitos**

O XML-ODBMS se propõe a resolver conflitos que existam entre o esquema do banco de dados de origem e destino. Assim deve ser utilizado um mecanismo que permita definir regras que indiquem como estas diferenças serão resolvidas. Este mecanismo é disponibilizado pela linguagem XML-ODMG.

- **Exportação de dados**

O XML-ODBMS deve exportar os dados de uma base de dados e organizá-los em um documento XML-ODMG.

- **Importação de dados**

O XML-ODBMS deve importar os dados que estão organizados no XML-ODMG de Dados para uma base de dados, respeitando as regras definidas para a resolução de conflitos.

- **Suporte a diferentes SGBDs**

O XML-ODBMS deve oferecer suporte a diferentes SGBDs para atender as organizações que provavelmente possuem diferentes tecnologias de banco de dados, como *Oracle*, *SQL Server*, *Informix*, *PostgreSQL* entre outros.

- **Suporte a diferentes plataformas**

Deve poder atuar sobre diferentes plataformas (*Windows, Linux, Unix, Solaris* entre outros) para os SGBDs das Organizações.

5.4 Requisitos Não Funcionais

São funcionalidades do XML-ODBMS cujas ausências não comprometem o objetivo principal da ferramenta. O intuito dessas funções é o de facilitar o uso do XML-ODBMS e compreendem:

- **Permitir a seleção mais customizada dos dados que serão exportados de uma base de dados para o documento XML-ODMG**

  O XML-ODBMS permite que os dados que serão exportados para documentos em XML-ODMG possam ser selecionados através do uso de sentenças SQL, para banco de dados relacionais e objeto-relacionais ou OQL para banco de dados orientados a objetos, oferecendo assim um alto nível de customização na seleção.

- **Permitir a seleção mais customizada dos dados organizados em XML-ODMG que serão importados para um base de dados**

  O XML-ODBMS também permite que os dados organizados em documentos em XML-ODMG possam ser selecionados através de sentenças XQL e para uma posterior importação.

- ***Interface* Gráfica**

  O XML-ODBMS oferece uma *interface* gráfica que tem o objetivo de facilitar transações de intercâmbio de dados.

5.5 Arquitetura do XML-ODBMS

A arquitetura do XML-ODBMS é uma evolução da arquitetura básica dos SIBD apresentada no capítulo 3, que busca oferecer uma maior flexibilidade e portabilidade na sua utilização.

Esta arquitetura atende os requisitos dipostos no capítulo 3 e é composta de três elementos que funcionam de forma integrada. São eles: **DBMS *Interface*, XML-ODMG *e* XML-ODBMS *Engine.***

A figura 5.1 mostra graficamente a arquitetura do XML-ODBMS e os relacionamentos entre seus elementos.

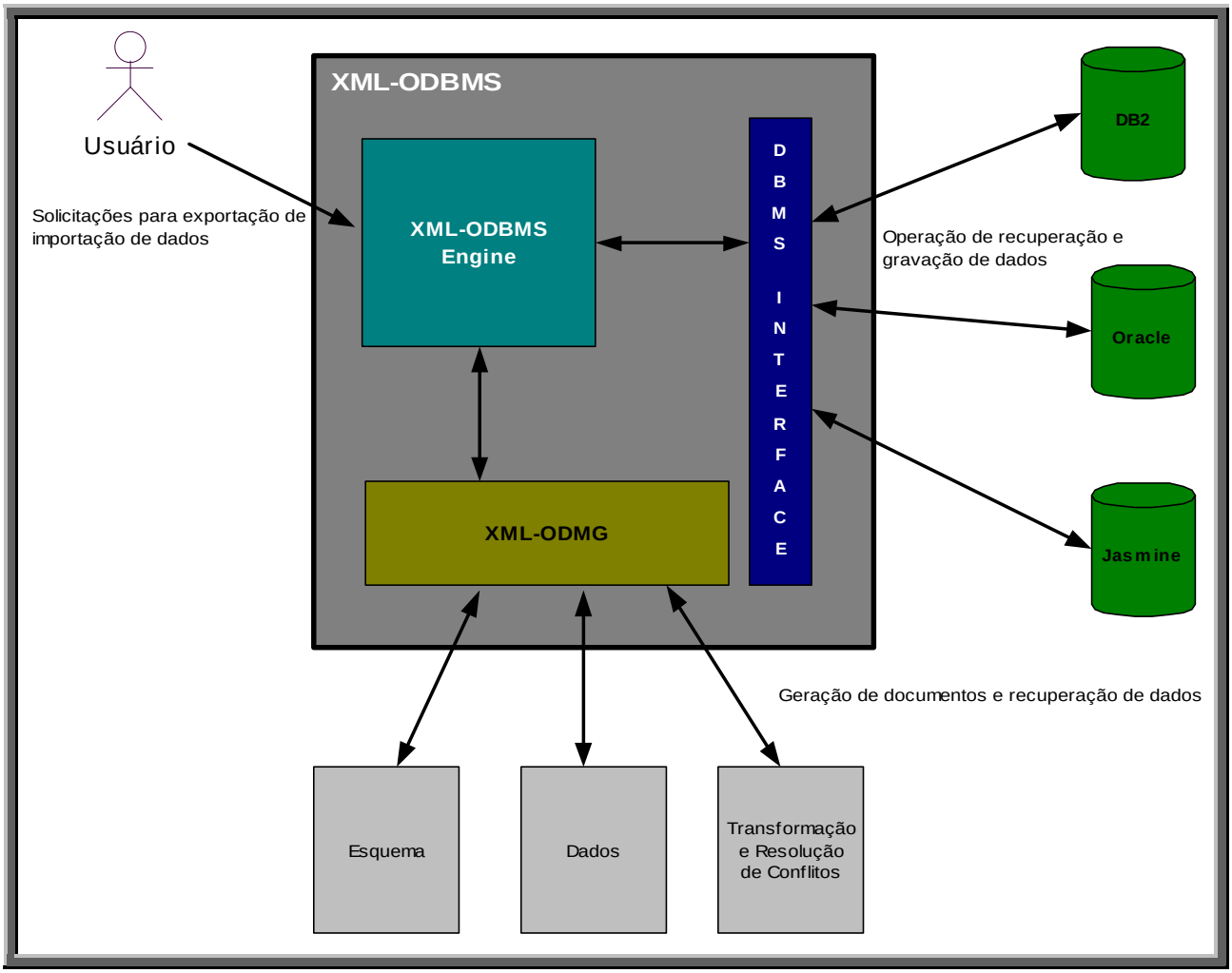

Figura 5.1: Arquitetura do *X*ML-ODBMS

DBMS *Interface*

Este componente especifica um conjunto de métodos que permite a interação com o SGBD utilizado, tanto na importação como na exportação. Quando um SGBD está envolvido em uma operação de intercâmbio de dados, os métodos definidos por esta *interface* devem ser implementados de forma a tratar as particularidades de cada SGBD. A implementação desta *interface* para um determinado SGBD é chamada de *driver* XML-ODBMS. Assim, para utilizar o SGBD *Oracle* deve-se possuir um *driver* XML-ODBMS para este SGBD.

Este elemento garante a transportabilidade do XML-ODBMS em relação aos SGBDs, uma vez que, a incorporação de um novo depende apenas da implementação destes métodos.

Os métodos definidos por esta *interface* são responsávies por:

- realizar a conexão com o SGBD;
- obter os dados do banco de dados, de acordo com os tipos existentes;
- incluir dados no banco de dados;

- executar sentenças SQL ou OQL definidas pelos usuários.

XML-ODMG

Este componente representa os documentos da linguagem XML-ODMG que serão utilizados para estruturar os dados extraídos do banco de dados e para definir as regras para a resolução de conflitos existentes entre os bancos de origem e destino.

XML-ODBMS *Engine*

Este componente é responsável por exportar os dados de um determinado banco de dados para um documento XML-ODMG e importá-los de um documento XML-ODMG para um outro banco de dados. O XML-ODBMS *Engine* vai utilizar os métodos definidos pelo *DBMS Interface* para obter os dados e organizá-los no documento XML-ODMG de Dados conforme as definições do XML-ODMG de Esquema. No momento da importação, o XML-ODBMS *Engine* utiliza-se novamente dos métodos do DBMS *Interface* para incluir os dados, fazendo uma análise da estrutura e aplicando as regras de resolução de conflitos definidas no XML-ODMG de Transformação de Resolução de Conflitos.

Os algorítmos que descrevem a interação destes três elementos de forma a realizar as operações de exportação e importação são apresentados nas figuras 5.2 e 5.3 respectivamente.

01	**Início**
02	Usuário solicita exportação dos dados
03	XML-DBMS Engine conecta-se como SGBD
04	XML-DBMS Engine consulta a base e obtém os dados
05	Enquanto houver dados recuperados do banco de dados
06	Interpreta o XML-ODMG de Esquema
07	Grava os dados corrente no XML-ODMG de Dados
08	**Fim**

Figura 5.2: Algoritmo de exportação de dados

01	**Início**
02	Usuário solicita a importação dos dados
03	XML-DBMS Engine conecta-se com o SGBD
04	XML-DBMS Engine obtém os dados contidos no XML-ODMG de Dados
05	Enquanto houver dados recuperados do XML-ODMG de Dados
06	Interpreta o XML-ODMG de Transformação e Resolução de Conflitos
07	Se houver conflito
08	Aplica as regras para resolução de conflitos presentes no XML-ODMG de Resolução de Conflitos
09	Inclui o dado corrente no banco de dados
10	**Fim**

Figura 5.3: Algorítmo de importação de dados

Os dois algorítmos iniciam-se com uma intervenção do usuário a fim de realizar uma solicitação para uma operação de importação ou exportação. Após esta solicitação o XML-DBMS *Engine* conecta-se ao SGBD utilizando os métodos definidos no DBMS *Interface.* Quando a conexão é concretizada os dois algorítmos obtêm os dados que serão exportados ou importados. Caso seja uma operação de exportação os dados são obtidos de uma base de dados localizado em um SGBD, caso contrário, os dados são obtidos de um documento XML-ODMG de Dados.

5.6 Implementação do XML-ODBMS

O XML-ODBMS foi desenvolvido em Java, cuja escolha é justificada pela transportabilidade por ela oferecida. Esta transportabilidade garante o atendimento a um dos principais requisitos de um SIBD, o qual concebe que um SIBD deve ser uma ferramenta independente da plataforma.

Os detalhes sobre a implementação desta ferramenta serão apresentados por componente da arquitetura XML-ODBMS (DBMS *Interface*, XML-ODMG e XML-ODBMS *Engine*), onde para cada um deles será mostrado o diagrama de classe, em notação UML, que conterá as classes que envolvem o respectivo componente da arquitetura.

5.6.1 DBMS *Interface*

A figura 5.4 mostra o diagrama de classe referente ao DBMS *Interface*.

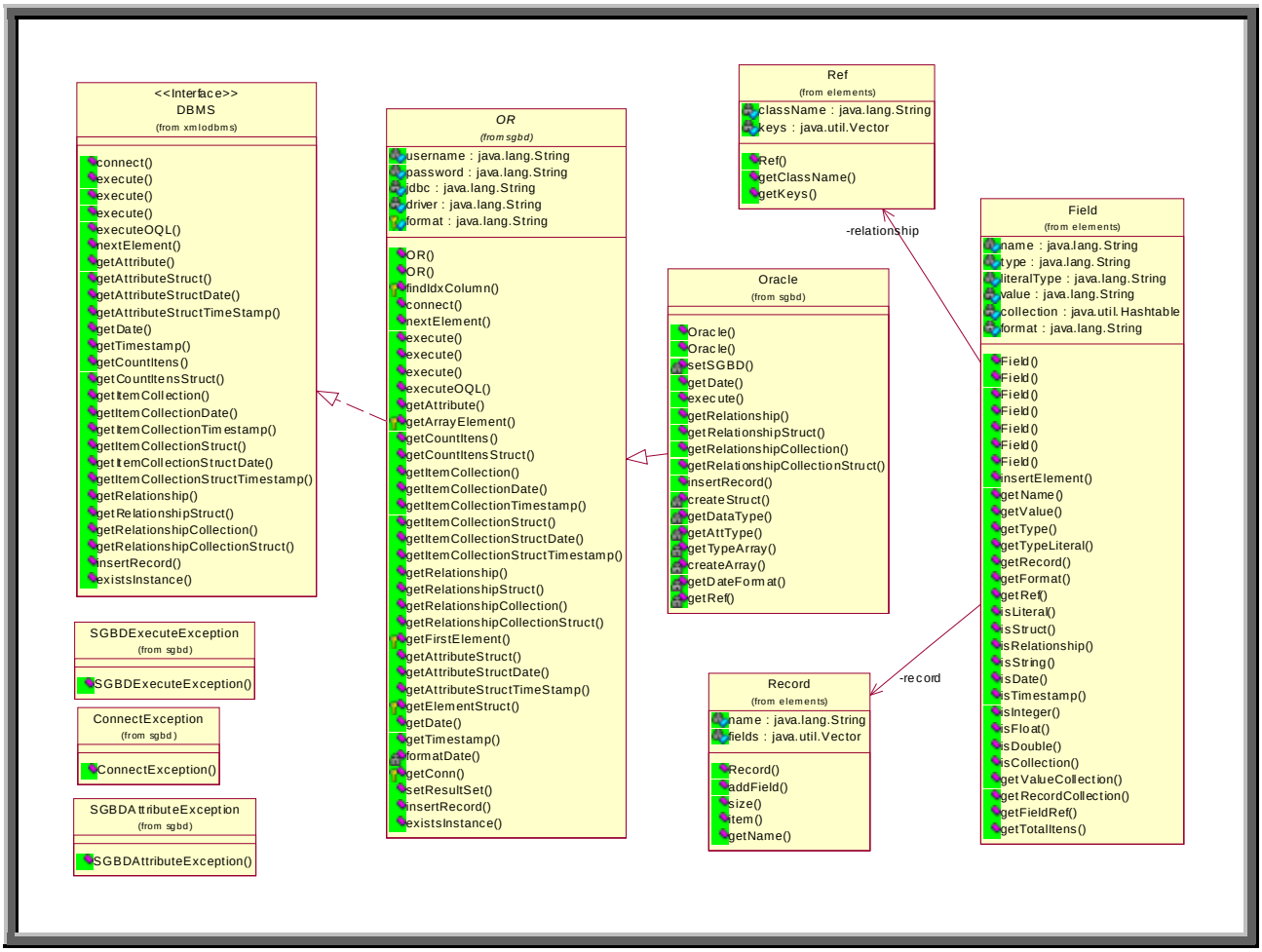

Figura 5.4: Diagrama de classe do DBMS *Interface*

A *interface* Java, `DBMS`, define os métodos para acesso e manipulação dos dados de um SGBD que esteja envolvido nas operações de exportação e/ou importação. Seus métodos definem funções de: conexão com o SGBD, execução de consultas, navegação sobre os resultados das consultas, obtenção e inclusão de dados.

O método `connect` realiza a conexão com o SGBD, já os métodos `execute` e `executeOQL` operam consultas a uma base de dados do SGBD. Ao utilizar o método `executeOQL`, deve ser informada uma sentença de consulta, seja OQL ou SQL, que será passada diretamente para o SGBD que fará a sua interpretação e execução retornando os valores selecionados ou avisando de erros na formação da sentença.

Quando é realizada uma consulta ao SGBD, pode-se recuperar vários valores como resultado, a *interface* DBMS permite que se navegue dentro deste conjunto de valores através do método `nextElement`, que posiciona o cursor de pesquisa no próximo elemento do conjunto de resultados. Os métodos *gets*, como `getAttribute`, `getDate`, `getItemCollection`, `getRelationship`, `getAttributeStruct` entre outros,

são métodos utilizados para recuperar o valor conforme o seu tipo. Por exemplo, o método `getDate` permite recuperar um valor do tipo data, enquanto o método `getCollection` um determindo valor de um conjunto.

A inclusão dos dados na base de dados do SGBD é realizada através do método `insertRecord`, que deve tratar uma estrutura de dados organizadas sobre as classes `Record`, `Field` e `Ref`. Onde a classe `Record` representa uma instância de classe que deve ser inserida no banco de dados e que possui um conjunto de objetos da classe `Field`, a qual vai representar o valor de cada atributo da instância que está sendo incluída no banco de dados. A classe `Ref` é utilizada como um atributo da classe `Field` e representa o valor de relacionamento com outra classe.

A classe `OR` é a implementação da *interface* `DBMS` para os SGBD objetos-relacionais. É uma classe abstrata que implementa as particularidades de um SGBD objeto-relacional como: detalhes da conexão JDBC e tratamento dos tipos de dados.

Esta classe foi implementada utilizando a especificação do JDBC 2.0, a qual oferece suporte aos elementos de um banco de dados objeto-relacional como, coleções, tabelas aninhadas, objetos entre outros, diferentemente da especificação 1.0 que tratava apenas modelos relacionais, como definido na especificação SQL3 [SUN 99]. Toda vez que um SGBD objeto-relacional for incluído no XML-ODBMS, a classe que o representa deve herdar da `OR` como acontece com a classe `Oracle` que representa o SGBD *Oracle 8i*. A dificuldade na implementação de um *driver* XML-ODBMS consiste em implementar os métodos *gets* e *insert* da *interface* `DBMS` fazendo com que estes obtenham os dados equivalentes no banco de dados. Por exemplo, o método `getItemCollection` retorna o valor de um determinado item em um coleção, na implementação do *driver* XML-ODBMS para *Oracle* este método deve recuperar um valor de um tipo de dado *VARRAY*[22] ou de uma *NestedTable.*[23]

O XML-ODBMS traz apenas o *driver* XML-ODBMS para o SGBD objeto-relacional *Oracle 8i,* o qual foi utilizado para realizar os testes da ferramenta. Para incorporar um outro SGBD objeto-relacional, orientado a objeto ou relacional deve ser implementado o *driver* XML-ODBMS respectivo.

[22] Tipo de dado exclusivo do Oracle 8i, que representa um vetor de valores
[23] Tipo de dado exclusivo do Oracle 8i, que representa um tabela como um campo de uma outra tabela.

5.6.2 XML-ODMG

Para apresentar o diagrama de classe deste item da arquitetura, dividiu-se este em duas partes: a primeira referente aos elementos referentes ao XML-ODMG de Esquema e de Dados (figura 5.5) e a segunda sobre os elementos que cuidam do XML-ODMG de Transformação de Resolução de Conflitos (figura 5.6).

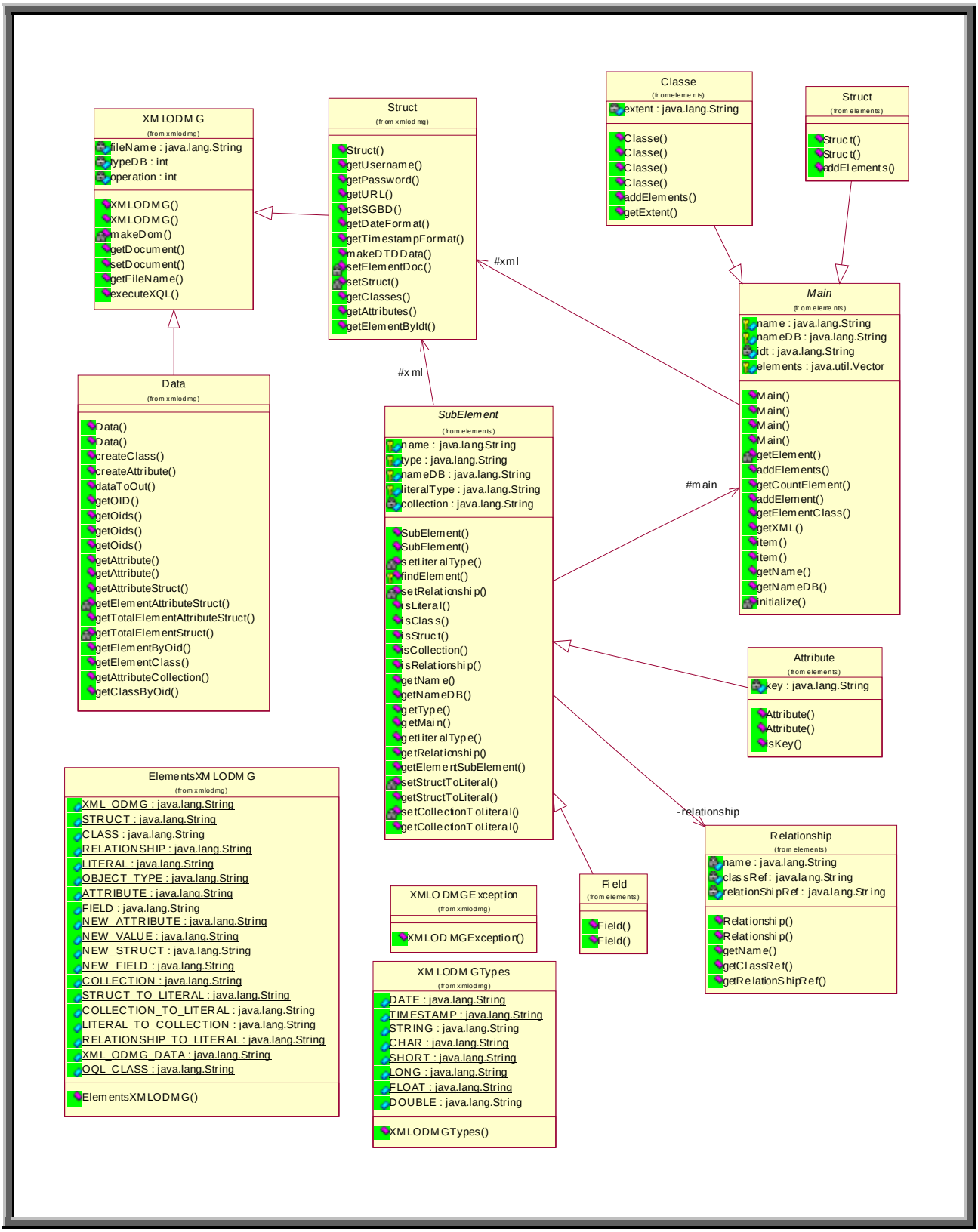

Figura 5.5: Diagrama de classe do XML-ODMG – Parte 1

Conforme o diagrama da figura 5.5, a classe XML-ODMG é a classe básica para a formação dos documentos XML-ODMG de Esquema, Dados e Transformação de Resolução

de Conflitos. Ela realiza todas as funções relativas ao tratamento de documentos XML, como submeter o documento à ação do *XML parser* e gerar uma árvore DOM deste documento. O método `executeXQL` realiza uma consulta ao documento XML através de uma sentença XQL, retornando uma outra árvore DOM como resultado.

O documento XML-ODMG de Esquema é representado pela classe `Struct`, que herda da classe XML-ODMG e disponibiliza métodos para obter informações contidas neste documento. O XML-ODMG de Dados é representado pela classe `Data` que também herda da classe XML-ODMG e possui métodos para criação de elementos que representam instâncias de classes (como `createClass` e `createAttribute`) e para recuperação de dados como `getAttribute`, que recupera o valor do atributo de uma instância, e `getAttributeCollection` que recupera o valor de um item de um atributo do tipo coleção.

As classes `Main`, `SubElement`, `Class`, `Struct`, `Attribute`, `Field` e `Relationship` representam os elementos do XML-ODMG de Esquema que estão representando a estrutura das classes, dos tipos estruturados, dos atributos, dos campos e dos relacionamentos respectivamente. Estas classes são utilizadas pelo XML-ODBMS para realizar a interpretação da estrutura montada dentro do XML-ODMG de Esquema a fim de obter os dados do banco de dados de origem e para organizá-los dentro do XML-ODMG de Dados.

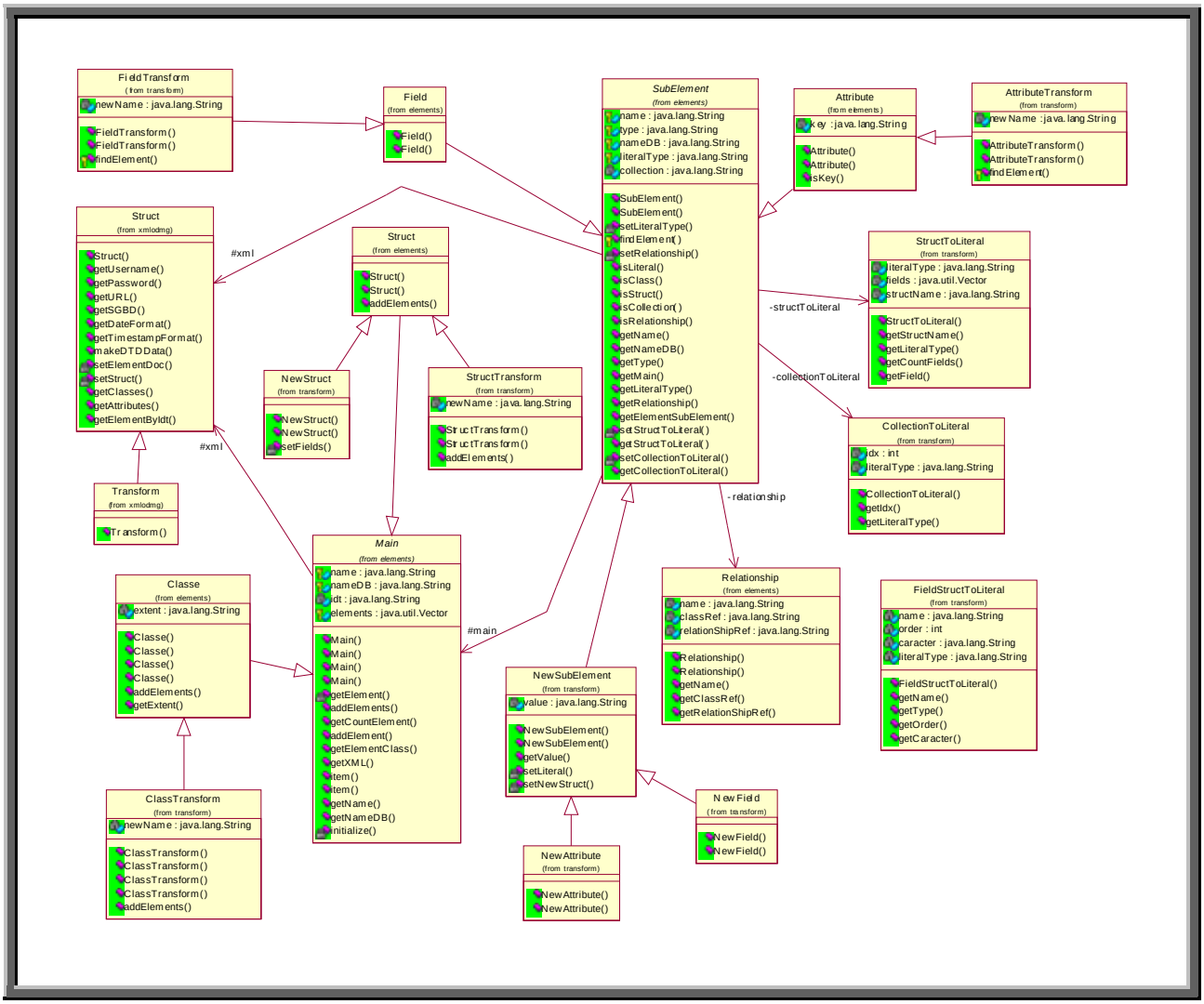

Figura 5.6: Diagrama de classe do XML-ODMG – Parte 2

Observando o diagrama da figura 5.6, a classe `Transform` representa o documento XML-ODMG de Transformação e Resolução de Conflitos, e herda da classe `Struct`. Esta herança é explicada pelo fato de que o XML-ODMG de Transformação e Resolução de Conflitos possui todos os elementos do XML-ODMG de Esquema. As classes `ClassTransform`, `StructTransform`, `AttributeTransform` e `FieldTransform` representam, como objetos Java, as classes, os tipos estruturados, os atributos das classes e os campos dos tipos estruturados respectivamente, que estão presentes no XML-ODMG de Transformação e Resolução de Conflitos. Através da interpretação destas classes o XML-ODBMS obtém os dados do XML-ODMG de Dados e os inclui no banco de dados.

Para resolver os conflitos o XML-ODBMS utiliza-se das classes `NewSubElement`, `NewAttribute`, `NewField`, `NewStruct`, `StructToLiteral`,

`FieldStructToLiteral` e `CollectionToLiteral` que representam, em objetos Java, as regras de resolução de conflitos definidas no XML-ODMG de Transformação de Resolução de Conflitos.

5.6.3 XML-ODBMS *Engine*

A figura 5.7 apresenta o diagrama de classe que contém as classes que formam o XML-ODBMS *Engine* e suas ligações com os outros elementos da arquitetura, DBMS *Interface* e XML-ODMG. O XML-ODBMS *Engine* compõe-se de duas classes: `DBMSToXML` reponsável por realizar a exportação dos dados para um XML-ODMG de Dados e `XMLToDBMS` que realiza a importação dos dados de um XML-ODMG de Dados para um banco de dados.

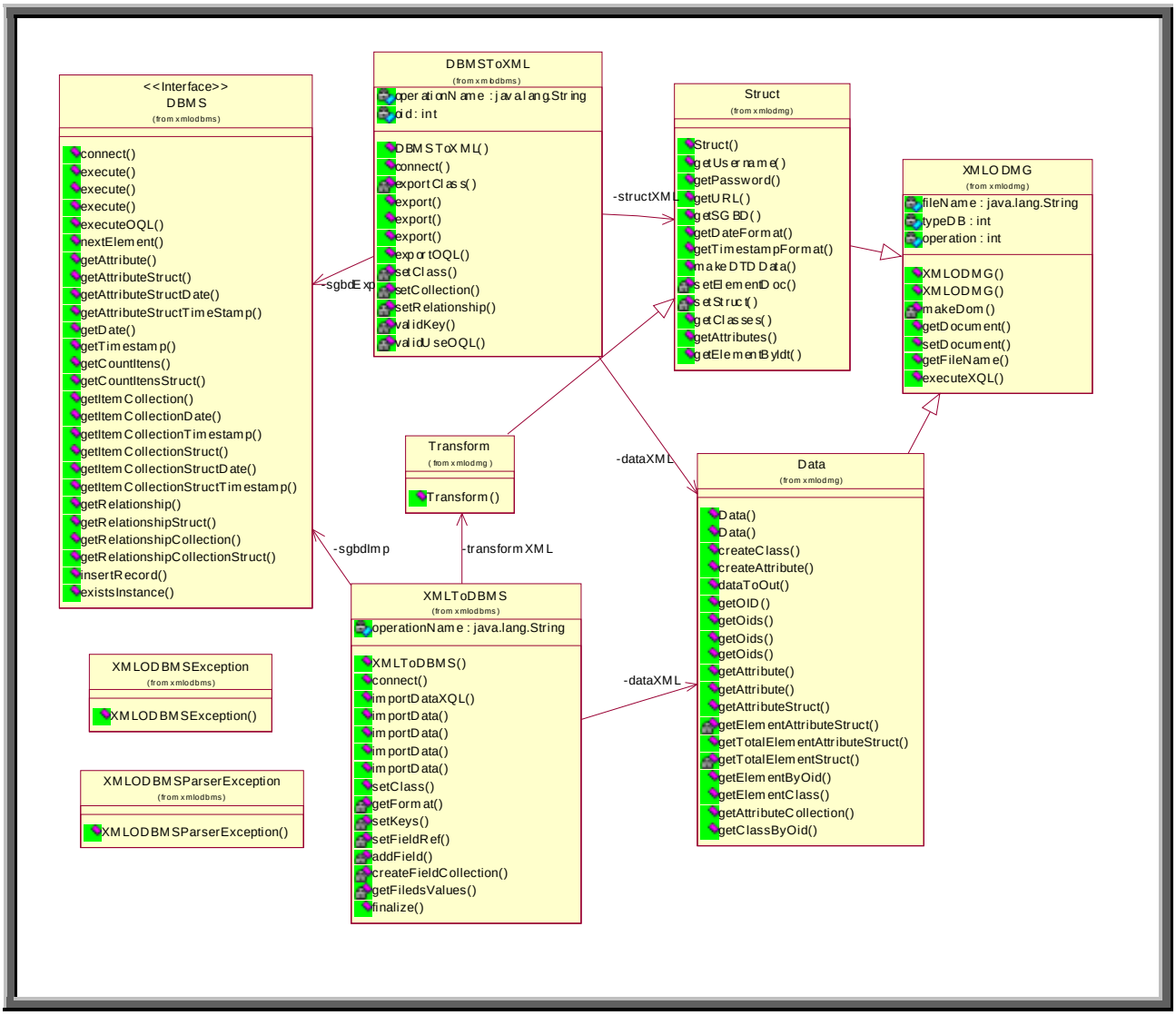

Figura 5.7: Diagrama de classe do XML-ODBMS *Engine*

Os métodos `export` e o método `exportOQL`, da classe `DBMSToXML`, utilizam os métodos da *interface* `DBMS` em conjunto com os métodos `setClass`, `setCollection` e `setRelationship` para recuperar os dados do banco de dados e organizá-los em um documento XML-ODMG de Dados através da utilização dos métodos da classe `Data`.

A classe `XMLToDBMS` dispõe dos métodos `importData` e `importDataXQL` que recuperam os dados do XML-ODMG de Dados, através dos métodos da classe `Data`, e os importam para o banco de dados utilizando os métodos da *interface* `DBMS`.

5.6.4 Algoritmos de Exportação / Importação

Nesta seção serão apresentados os algoritmos básicos utilizados pelo XML-ODBMS *Engine* para realizar as operações de exportação e importação, através das figuras 5.8 e 5.9 respectivamente.

Exportação

```
1  Inicio
2    /* Definição do procedimento de exportação dos dados do banco de dados
3       para o documento  XML-ODMG */
4   procedimento exportar( dados, driver, classe )
5   Inicio
6    /* Obtém um conjunto representando as instâncias da classe passada como
7       parâmetro  */
8    conjunto = driver.consultaSGBD( classe );
9
10   /* Exporta todos os valores retornados na consulta ao SGBD */
11   enquanto conjunto.existirElemento faça
12     valor = conjunto.obterValor
13     se valor = Relacionamento então
14       /* Chamada recursiva ao procedimento de exportação, a fim de obter os
15          valores das classes  que se relacionam com o valor corrente */
16       exportar( dados, driver, valor.relacionamento.classe );
17     senão
18       /* Grava o valor do documento XML-ODMG de Dados*/
20       dados.gravarValor( valor )
21     fim_se
22     conjunto.proximoElemento
23   fim_enquanto
24  Fim
25  /* Fim da definição do procedimento de exportação */
26
27 /* Início da análise dos documentos XML-ODMG */
28  esquema = obter_XML-ODMG_Esquema
29  dados   = obter_XML-ODMG_Dados
30  driver   = obter_driver_XML-ODBMS
31  /* Interpreta cada elemento (classe) representado no documento
32     XML-ODMG de Esquema */
33  enquanto esquema.existirElementos faça
34   /* Solicita a exportação da instância de uma determinada classe, fazendo chamada
35      ao procedimento de exportar */
36   exportar( dados, driver, esquema.classe )
37   esquema.proximoElemento
38  fim_enquanto
39 Fim
```

Figura 5.8: Algoritmo de exportação do XML-ODBMS *Engine*

A linha 4, do algoritmo da figura 5.8, define o procedimento de exportação, que tem a função de recuperar as informações de um banco de dados e gravá-las no documento XML-ODMG de Dados. Na linha 8 é realizada uma consulta ao banco de dados utilizando-se o *driver* XML-ODBMS disponível a fim de recuperar as instâncias da classe passada como argumento. A consulta realizada na linha 8 vai retornar um conjunto de valores que representam as instância de um determinada classe. Na linha 12 serão obtidos os dados deste conjunto que serão gravados no XML-ODMG de Dados através da instrução da linha 15. Caso o valor indique um relacionamento com outra classe, é feita uma chamada recursiva ao procedimento `exportar`, na linha 16, com o objetivo de exportar os valores da classe relacionada a fim de garantir integridade dos dados organizados no documento XML-ODMG de Dados. A linha 28 inicia-se a fase do algoritmo que obtém os documentos necessários, XML-ODMG de Esquema e XML-ODMG de Dados, linhas 28 e 29 respectivamente, como também o *driver* XML-ODBMS e inicializa o processo de exportação. As linhas 33, 36, 37 e 38 fazem a análise dos elementos incluídos no documento XML-ODMG de Esquema e faz chamadas ao procedimento de exportação para cada classe encontrada.

Importação

```
1  Inicio
2   /* Procedimento que realiza a importação dos dados de um documento XML-ODMG
3      de  Dados para um banco de dados */
4   procedimento importar(dados. driver, classe )
5   Inicio
6    /* Realiza uma consulta ao documento XML-ODMG de Dados a fim de obter os
7       dados que serão importados para um banco de dados. É retornado um conjunto
8       contendo os dados  originários do documento de Dados */
9    conjunto = dados.consulta_XML-ODMG-Dados( classe )
10
11   enquanto conjunto.existeElemento faça
12     valor = conjunto.obterValor
13     se valor = Relacionamento então
14      /* Chamada recursiva a fim de importar os valores da classe que
15          se está relacionando*/
16        importar( dados, driver, valor.relacionamento.classe );
17      senão
18        /* Caso existe um conflito, o valor obtido é alterado de forma que o conflito
20           seja resolvido */
21        se classe.existeConflito então
22          valor = classe.resolverConflito( valor )
23        fim-se
24        /* Insere o valor do banco de dados*/
25       driver.inserirValor( valor );
26     fim_se
27     conjunto.proximoElemento
28   fim_enquanto
29  Fim
30  /* Fim do procedimento de importação */
31
32  /* Início da análise dos documentos XML-ODMG */
33  transformacao = obter_XML-ODMG-Transformacao
34  dados = obter_XML-ODMG-Dados
35  driver = obter_driver_XML-ODBMS
36  /* Interpreta cada elemento (classe) do XML-ODMG de Transformação */
37  enquanto transformacao.existirElementos faça
38
39   /* Importação das instância da classe passada como parâmetro */
40   importar( dados, driver, transformacao.classe )
41   transformacao.proximoElemento
     fim_equanto
    Fim
```

Figura 5.9: Algoritmo importação do XML-ODBMS *Engine*

A linha 4, do algoritmo da figura 5.9, define o procedimento de importação responsável por obter os dados do XML-ODMG de Dados e importá-los para o banco de

dados. Na linha 9 é feita uma consulta ao XML-ODMG de Dados a fim de recuperar os dados que serão importados, retornando um conjunto contendo estes valores. Caso um determinado valor deste conjunto indique um relacionamento com outra classe, é realizada uma chamada recursiva ao procedimento de importação, na linha 16, a fim de importar os valores da classe relacionada, com o objetvio de manter integridade na base de dados. A linha 21 verifica se existe algum conflito definido, caso existe este é resolvido e o valor é gravado no banco de dados, do contrário a importação é suspensa e uma mensagem de erro é exibida. Na linha 32, obtém-se os documentos XML-ODMG necessários na operação, XML-ODMG de Transformação e Resolução de Conflitos e XML-ODMG de Dados, linhas 33 e 34 respectivamente, como também o *driver* XML-ODBMS que será utilizado, na linha 35, e inicializa o processo de importação. Entre as linhas 37 e 40 é feita à análise do documento XML-ODMG de Transformação e Resolução de Conflitos em conjunto com chamadas ao procedimento de importação para cada elementos analisado no XML-ODMG de Transformação e Resolução de Conflitos.

5.7 Restrições na utilização de herança no XML-ODBMS

O XML-ODBMS permite que instâncias de classes definidas através da herança de outras classes sejam exportadas e importadas, porém existem algumas restrições quanto ao uso desta funcionalidade. Em relação à exportação o XML-ODBMS não exportará os dados relativos a classes que possuam classes filhas. Assim qualquer classe que seja definida como pai de outra classe não será exportada para o documento XML-ODMG de Dados. Consequentemente a importação será feita apenas para classes que não possuam classes filhas. Caso alguma instância de uma classe exportada do modelo de origem seja equivalente a alguma classe no modelo de destino onde esta possua classes filhas, configura-se um conflito a nível de herança que não é resolvido através da XML-ODMG de Transformação e Resolução de Conflitos inviabilizando a operação de intercâmbio de dados.

5.8 Utilizando o XML-ODBMS

O XML-ODBMS dispõe de dois modos de utilização. O primeiro é voltado para os desenvolvedores de sistemas, que irão utilizar a API XML-ODBMS dentro de suas aplicações

a fim de realizar o intercâmbio de dados. O segundo é a utilização da *interface* gráfica para operar o intercâmbio de dados sem a necessidade de desenvolver aplicações para este fim.

5.8.1 Utilizando a API XML-ODBMS

A utilização da API XML-ODBMS se aplica quando desenvolvedores de aplicações têm a necessidade de incorporar em seus sistemas funcionalidades de intercâmbio de dados. XML-ODBMS permite que sua API possa ser utilizada dentro do código de diversas aplicações, fazendo com que estas possam realizar operações de exportação e importação de dados de acordo com suas necessidades.

Incorporar os recursos de importação e exportação nas aplicações consiste na utilização das classes `DBMSToXML` e `XMLToDBMS` respectivamente localizadas no pacote de classes Java `xmlodbms.jar`, disponibilizado pelo XML-ODBMS.

Exportação

A fim de oferecer uma maior flexibilidade ao desenvolvedor o XML-ODBMS, através da classe `XMLToDBMS` dispõe de quatro possibilidades de exportação:

1. exportar todas as instâncias de uma classe;
2. exportar instâncias filtrando por um valor de um determinado atributo da classe;
3. exportar todas as instâncias de um conjunto de classes;
4. exportar os dados através de uma setença SQL ou OQL.

A escolha por uma delas depende da solução encontrada por cada programador para atender suas necessidades.

Utilizar estas quatros alternativas de exportação consiste na chamada de quatros diferentes métodos da classe `XMLToDBMS`, que foram apresentados na seção 5.6, quando tratamos das classes do XML-ODBMS.

O código Java abaixo realiza a exportação dos dados de uma base de dados do SGBD objeto-relacional *Oracle 8i* para um documento XML-ODMG de Dados que é representado pelo arquivo `exportacao.xml`.

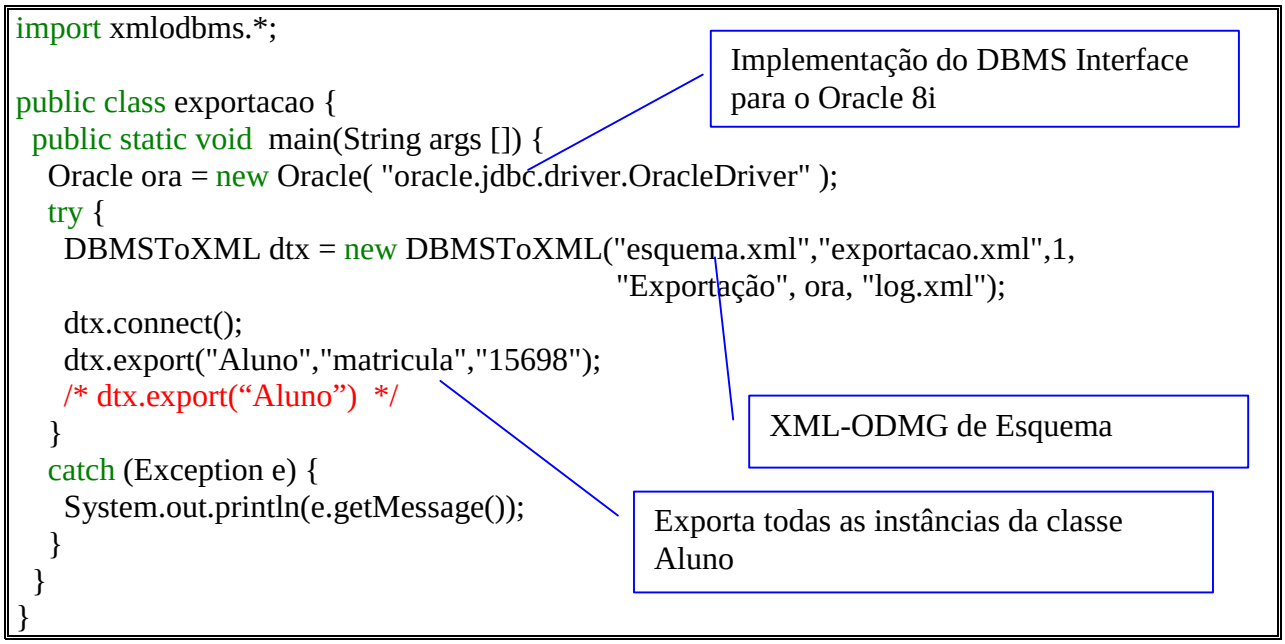

Figura 5.10: Exemplo de código Java para exportação de dados utilizando XML-ODBMS

No código Java da figura 5.10, o XML-ODBMS exporta do banco de dados todas as instâncias da classe `Aluno` cujo valor do atributo `matricula` seja igual 15698, através do método `export`.

Já o exemplo da figura 5.11 realiza a exportação de todas as instâncias das classes `Aluno` e `Professor`. Estas devem ser agrupadas em um vetor e este passado como parâmetro para o método `export`.

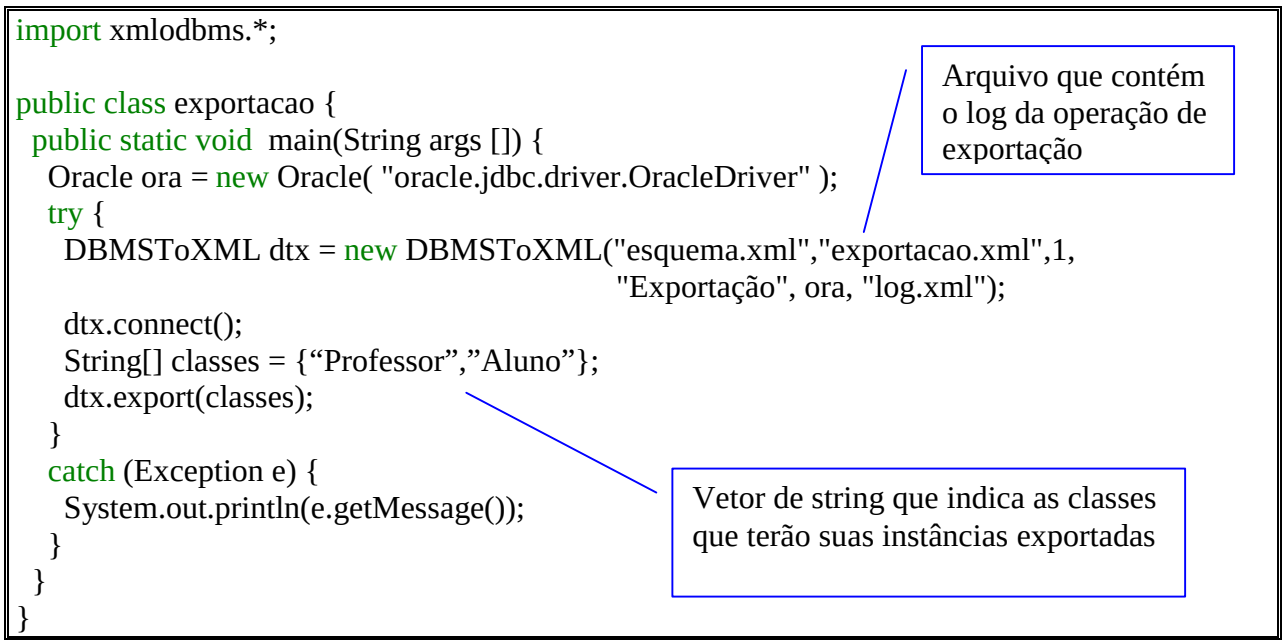

Figura 5.11: Exemplo de código Java para exportação de dados de mais de uma classe utilizando XML-ODBMS

Uma outra maneira de realizar a exportação é utilizar sentenças SQL ou OQL para recuperar os dados. Esta exportação é realizada através do método `exportOQL`, onde a sentença de consulta é passada como parâmetro. O exemplo da figura 5.12 ilustra este tipo de exportação.

```
import xmlodbms.*;

public class exportacao {
  public static void  main(String args []) {
    Oracle ora = new Oracle( "oracle.jdbc.driver.OracleDriver" );
    try {
      DBMSToXML dtx = new DBMSToXML("esquema.xml","exportacao.xml",1,
                                    "Exportação", ora, "log.xml");
      dtx.connect();
      String oql = “select a.matricula, a.pessoa.nome nome from alunos a where “ +
                   “a.pessoa.nome like \‘%JOSE%\’”;
      dtx.exportOQL(oql);
    }
    catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Recupera a matrícula e nome de todos os alunos que possuem a palavra Jose na composição de seu nome

Figura 5.12: Exemplo de código Java para exportação de dados através de sentenças SQL ou OQL

Para realizar a importação, definida no código Java da figura 5.12, é necessário que no XML-ODMG de Esquema exista uma classe de nome “OQLClass” que deve conter as propriedades dos atributos selecionados na sentença como mostra o exemplo da figura 5.13.

```
<class name=”OQLClass” idt=”oqlclass” nameDB=”OQLClass” extent=”OQLClass>
   <attribute name=”matricula” nameDB=”MATRICULA” key=”yes” type=”literal”
              length=”10” >
      <literal type=”short”/>
   </attribute>
   <attribute name=”nome” nameDB=”NOME”  type=”literal” length=”80”>
      <literal type=”string”/>
   </attribute>
</class>
```

Campo selecionado na sentença de consulta

Campo selecionado na sentença de consulta

Figura 5.13: Definição da classe OQLClass

Importação

Como na exportação, o XML-ODBMS também disponibiliza formas diferentes de importação:

1. importar todas as instâncias de um classe;
2. importar todas as instâncias de um classe filtrando pelo valor de um determinado atributo;
3. importar todas as instâncias de um conjunto de classes;
4. importar os dados selecionados através de uma sentença XQL.

Para cada um destes tipos de importação, o XML-ODBMS disponibiliza um método que pode ser acessado através da classe `XMLToDBMS`. O exemplo abaixo realiza a importação dos dados contidos em um XML-ODMG de Dados, representado pelo arquivo `exportacao.xml`, para um banco de dados de um SGBD objeto relacional *Oracle 8i*.

```
public class importacao {
  public static void  main(String args []) {
    Oracle ora = new Oracle( "oracle.jdbc.driver.OracleDriver" );
    try {
      XMLToDBMS xtb = new XMLToDBMS("transformacao.xml", exportacao.xml", 1,
                                    "Importação",ora," log.xml");
      xtb.connect();
      xtb.importData("Aluno");
      /* xtb.importData(“Aluno”,”matricula”,”12569”) */
     }
    catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

XML-ODMG de Transformação e Resolução de Conflitos

Importará todas as instâncias da classe Aluno cujo valor do atributo matricula seja igual a 12569

Figura 5.14 : Exemplo de código Java para importação de dados utilizando XML-ODBMS

O código Java da figura 5.14 importará todas as instâncias da classe `Aluno` presentes no documento XML-ODMG de Dados.

Para importar todas as instâncias de um conjunto de classes utiliza-se o mesmo príncipio da exportação. As classes são agrupadas em um vetor, que é passado como parâmetro, para o método `importData`, onde este importará todas as instâncias das classes

deste vetor. O código da figura 5.15 importa todas as instâncias das classes `Aluno`, `Professor` e `Disciplina`.

```
public class importacao {
  public static void  main(String args []) {
    Oracle ora = new Oracle( "oracle.jdbc.driver.OracleDriver" );
    try {
      XMLToDBMS xtb = new XMLToDBMS("transformacao.xml", exportacao.xml", 1,
                                                    "Importação",ora," log.xml");
      xtb.connect();
      String classes[] = {“Aluno”, “Disciplina”, “Professor”);
      xtb.importData(classes);
     }
    catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Vetor de classes

Figura 5.15: Exemplo de código Java para importação das instâncias de mais de uma classe

A fim de oferecer uma maior flexibilidade ao desenvolvedor, o `XML-ODBMS` permite selecionar os dados que serão importados do XML-ODMG de Dados através de sentenças XQL utilizando o método `importDataXQL`. O exemplo da figura 5.16 mostra este tipo de importação.

```
public class importacao {
  public static void  main(String args []) {
    Oracle ora = new Oracle( "oracle.jdbc.driver.OracleDriver" );
    try {
      XMLToDBMS xtb = new XMLToDBMS("transformacao.xml", exportacao.xml", 1,
                                                    "Importação",ora," log.xml");
      xtb.connect();
      xtb.importDataXQL(“//Aluno[pessoa/@nome='Jose Carlos’]”);
     }
    catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Sentença XQL que recupera todos os alunos com o nome de Jose Carlos

Figura 5.16: Exemplo de código Java para importação através de um sentença XQL

5.8.2 *Interface* Gráfica

O XML-ODBMS disponibiliza uma *interface* gráfica que possibilita ao usuário realizar exportações e importações. O objetivo principal da *interface* gráfica é dispor de uma alternativa de intercâmbio de dados mais simples.

A figura 5.8 mostra a tela de apresentação da *interface* gráfica do XML-ODBMS.

Figura 5.17: Tela de apresentação da *interface* gráfica

Na tela de apresentação o usuário escolhe se deseja realizar uma exportação ou uma importação de dados. Para cada uma destas opções existe uma tela específica que solicita informações necessárias para realizar cada operação. A figura 5.9 apresenta a tela de exportação do XML-ODBMS.

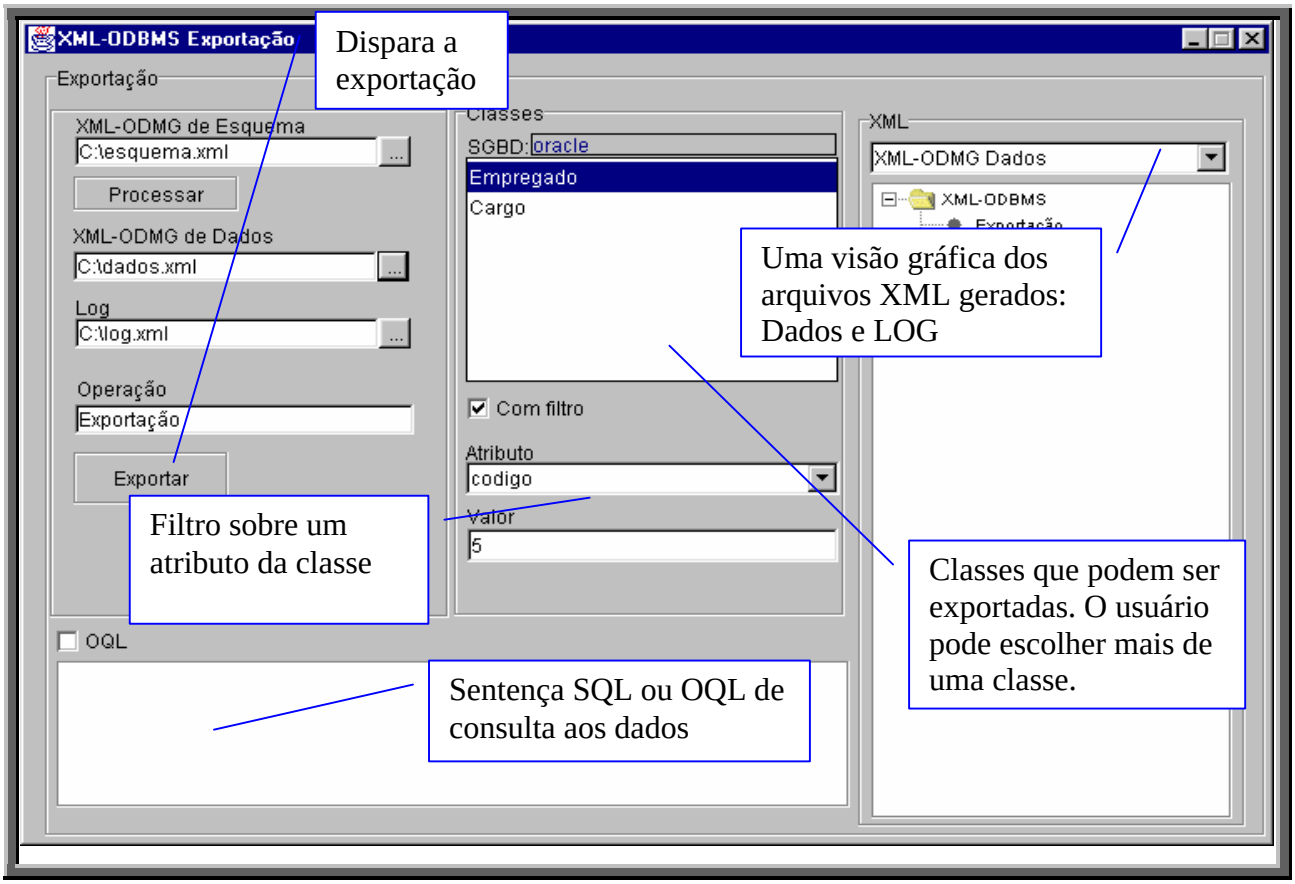

Figura 5.18: Tela de exportação da *interface* gráfica

A tela de importação é apresentada na figura 5.19.

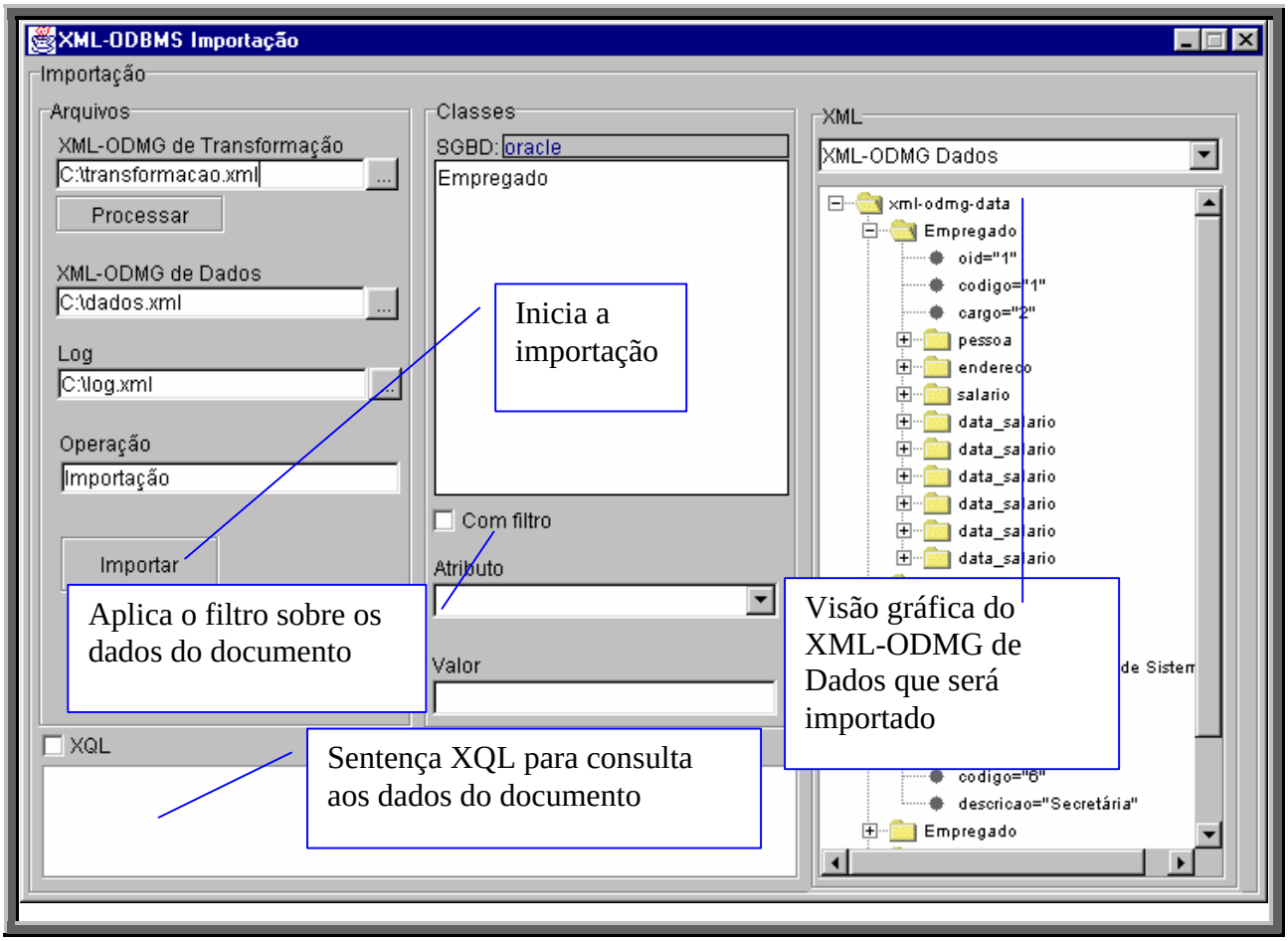

Figura 5.19: Tela de importação da *interface* gráfica

5.9 Conclusão

O XML-ODBMS é uma proposta inovadora de SIBD, que oferece uma plataforma de suporte aos SGBDs orientados a objetos e objeto-relacionais, potencial ainda inexplorado pelos SIBDs atuais. Por ser uma ferramenta desenvolvida em linguagem Java, garante a transportabilidade necessária para o sucesso do SIBD.

Sua arquitetura, formada pelos elementos DBMS *Interface*, XML-ODMG e XML-ODBMS *Engine*, garante flexibilidade nas operações de exportação e importação.

O DBMS *Interface* permite que um novo SGBD seja facilmente acoplado ao XML-ODBMS aumentado o seu universo de utilização, o XML-ODMG oferece flexibilidade para definir os esquemas de dados, organizá-los e definir regras para a resolução de conflitos e o XML-ODBMS *Engine* disponibiliza alternativas de exportação e importação conforme as necessidades do usuário.

XML-ODBMS permite que suas funções de exportação e importação possam ser utilizadas de duas formas: a primeira através da utilização das classes Java de exportação e importação, `DBMSToXML` e `XMLToDBMS` respectivamente, permitindo desenvolver aplicações com recursos de intercâmbio de dados. A segunda, através de uma *interface* gráfica que permite ao usuário operar exportações e importação de dados sem a necessidade de desenvolver aplicações.

Dentre as características do XML-ODBMS, apresentadas neste capítulo, devem ser destacadas o suporte a banco de dados orientado a objetos e objeto-relacional, a resolução de conflitos e a possibilidade de acoplamento de novos SGBDs, como pontos diferenciais em relação a outros SIBDs.

Capítulo 6

Um Estudo de Caso com o SIBD XML-ODBMS

6.1 Introdução

Este capítulo tem como objetivo mostrar uma situação prática de utilização do XML-ODBMS. Para isso vão ser utilizados dois modelos de dados (origem e destino), implementados no SGBD objeto-relacional *Oracle 8i*.

A figura 6.1 mostra o XML-ODBMS atuando sobre os bancos de dados de origem e destino, a partir da definição dos documentos XML-ODMG de Esquema e de Transformação e Resolução de Conflitos e do uso do *driver* XML-ODBMS para *Oracle 8i*.

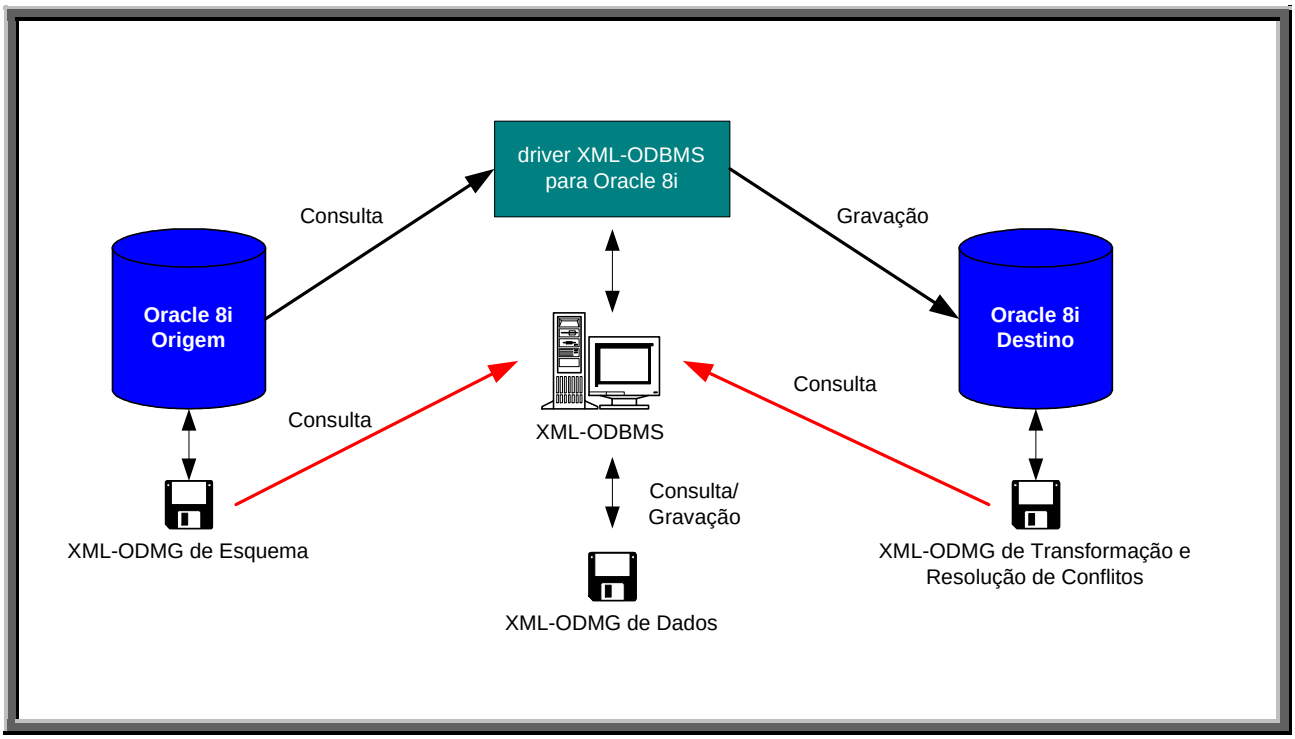

Figura 6.1: XML-ODBMS na aplicação-exemplo

6.2 Definição dos documentos XML-ODMG

A definição dos documentos em XML-ODMG consiste no processo de mapeamento dos objetos de um banco de dados objeto-relacional para os elementos do XML-ODMG, fazendo com que o XML-ODBMS, através dos métodos do DBMS *Interface*, seja capaz de utilizar os documentos XML-ODMG para exportar e importar os dados.

A relação entre os elementos da XML-ODMG e os objetos do SGBD é feita no momento da implementação do *driver* XML-ODBMS e deve ser seguida pelo usuário que vai definir os documentos XML-ODMG de Esquema e de Transformação e Resolução de Conflitos. Cada *driver* XML-ODBMS pode ter uma maneira diferente de fazer a relação entre os elementos, uma vez que nem sempre os SGBDs disponibilizam os mesmos tipos de dados e os mesmos recursos de implementação. Por exemplo, um atributo do tipo coleção definido em XML-ODMG pode ser representado de maneiras diferentes nos SGBDs objetos-relacionais *Oracle* e *PostgreSQL*.

Nesta demonstração será utilizado o *driver* XML-ODBMS para Oracle desenvolvido neste trabalho junto com o XML-ODBMS.

6.2.1 Definição do modelo de origem

O modelo de dados de origem é constituído de cinco entidades, `Empregados`, `Cargos`, `Pessoa`, `Salario` e `Endereco` que estão inseridas no modelo conceitual da figura 6.2:

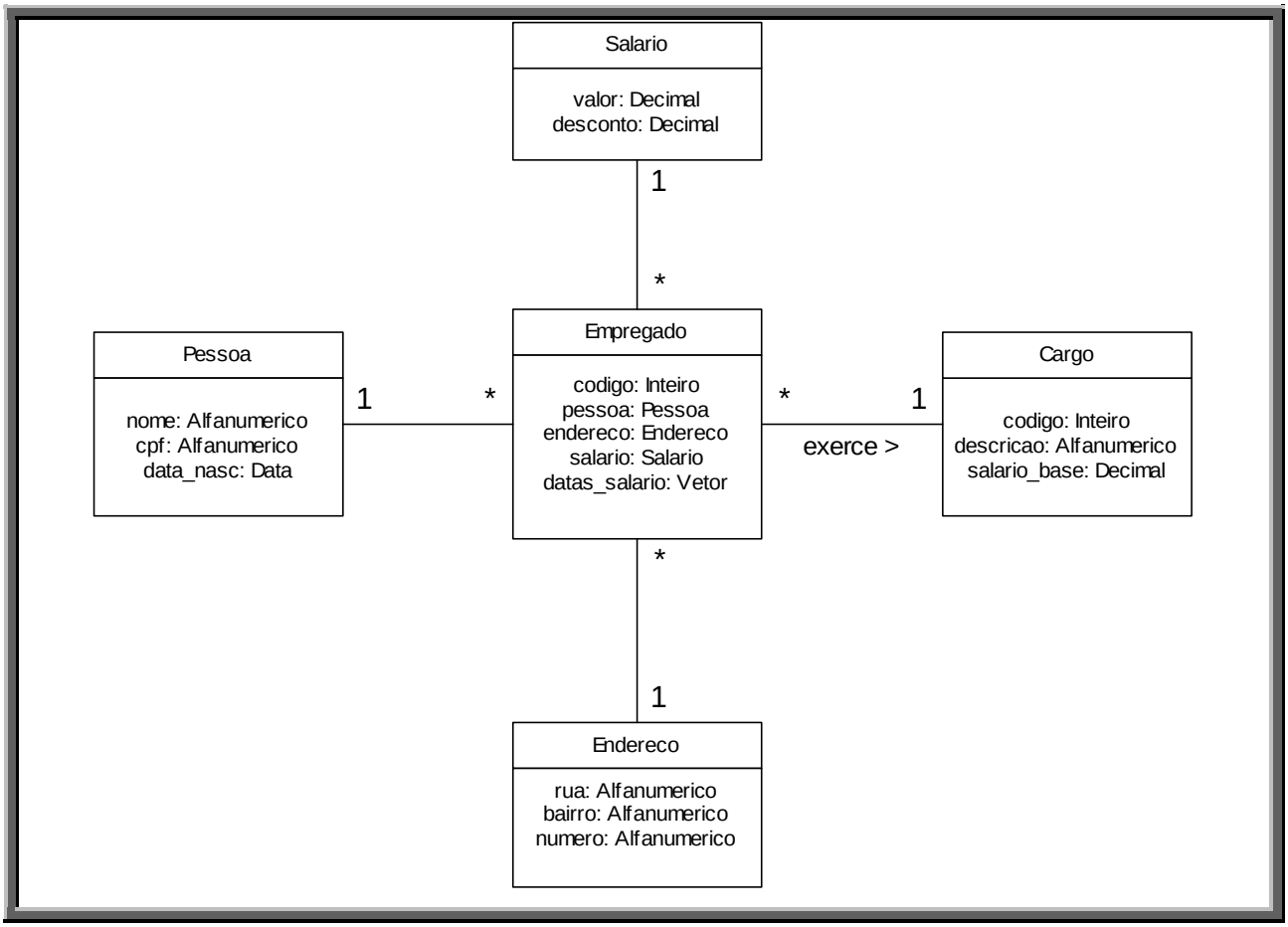

Figura 6.2: Definição conceitual das entidades do modelo de origem

Antes de criar a representação das entidades no XML-ODMG de Esquema, deve-se definir o cabeçalho do documento, ou seja, criar o elemento raiz `xml-odmg`.

```
<xml-odmg username="origem" urlJDBC="jdbc:oracle:thin:@servidor:1521:oracle"
          typeDB="OBJECT-RELATIONAL" timestampFormat="HH:mm:ss"
          dbms="oracle" password="origem" dateFormat="dd/MM/yyyy">
...
</xml-odmg>
```

Figura 6.3: Definição do cabeçalho do documento XML-ODMG de Esquema

A figura 6.3 apresenta a definição do cabeçalho do documento XML-ODMG de Esquema que contém as informações necessárias para realizar a conexão com o banco de dados, como `username` e `password` determinando respectivamente o usuário e a senha que permitem acesso ao SGBD. O atributo `urlJDBC` informa a *string* de conexão que deve ser utilizada pela implementação do DBMS *Interface* para realizar a conexão como SGBD.

Em relação à crição das entidades `Cargos` e `Empregados` no *Oracle*, ambas serão implementadas como *object-tables* , como mostra o *script* da figura 6.4.

```
/* Definição do tipo que representa uma classe */
create or replace type T_Cargo as object (
  codigo number(4),
  descricao varchar2(100),
  sal_base number(10,2)
)
/

/* Definição da object-table, que representa as instâncias de um classe */
create table cargos of T_Cargo (
  codigo not null primary key
)
/
```

Figura 6.4: *Script* de criação das *objects-tables* do esquema de destino

```
<class nameDB="T_CARGO" name="Cargo" idt="cargo" extent="CARGOS">
  <attribute type="literal" nullable="no" nameDB="CODIGO" name="codigo" key="yes"
             length=”4”>
    <literal type="short"/>
  </attribute>
  <attribute type="literal" nullable="no" nameDB="DESCRICAO" name="descricao"
             length=”80”>
    <literal type="string"/>
  </attribute>
  <attribute type="literal" nullable="no" nameDB="SAL_BASE" name="salario_base"
             length=”8.2” >
    <literal type="float"/>
  </attribute>
</class>
```

Figura 6.5: Definição da classe `Cargo,` em XML-ODMG, do esquema de origem

Na definição do XML-ODMG de Esquema presente na figura 6.5, a entidade `Cargo` é representada como uma classe de nome “Cargo”, e equivale ao elemento `T_CARGO`, no banco de dados. A *object-table* que armazena as tuplas da entidade `Cargo` é indicada pelo atributo `extents` no XML-ODMG. Desta forma toda vez que for solicitada uma exportação da classe `Cargo`, a implementação do DBMS *Interface* deve consultar a *object-*

table CARGOS e retornar cada tupla deste objeto como se fosse uma instância da classe Cargo.

A entidade Empregado deve ser definida da mesma forma, tanto no XML-ODMG como no banco de dados. Mas existem algumas particularidades que devem ser tratadas antes da sua definição. Os atributos, Pessoa, Endereco e Salario são formados por uma estrutura individual que envolvem outros atributos. Estes tipos de atributos são tratados como tipos estruturados no Oracle. O *script* de criação destes objetos mostrado na figura 6.6.

```
/* Definição do tipo que representa o atributo Endereço */
create or replace type T_Endereco as object (
  rua    varchar2(100),
  bairro varchar2(100),
  numero varchar2(10)
)
/

/* Definição do tipo que representa o atributo salário */
create or replace type T_Salario as object (
  valor    number(10,2),
  desconto number(10,2)
)
/

/* Definição do tipo que representa o atributo do tipo Pessoa */
create or replace type T_Pessoa as object (
  nome varchar2(80),
  cpf  char(11),
  dat_nasc date
)
/
```

Figura 6.6: *Script* de criação dos tipos estruturados do esquema de destino

Os objetos T_ENDERECO, T_SALARIO e T_PESSOA são tipos estruturados definidos pelo usuários que podem ser utilizados na definição do domínio de qualquer atributo. A figura 6.7 mostra a definição, em XML-ODMG, destes elementos.

```
<struct nameDB="T_ENDERECO" name="Endereco" idt="endereco">
  <field type="literal" nullable="no" nameDB="RUA" name="rua" length=”80”>
    <literal type="string"/>
  </field>
  <field type="literal" nullable="no" nameDB="BAIRRO" name="bairro" length=”80”>
    <literal type="string"/>
  </field>
  <field type="literal" nullable="no" nameDB="NUMERO" name="numero" length=”5”>
    <literal type="string"/>
  </field>
</struct>
<struct nameDB="T_SALARIO" name="Salario" idt="salario">
  <field type="literal" nullable="no" nameDB="VALOR" name="valor" length=”8.2”>
    <literal type="float"/>
  </field>
  <field type="literal" nullable="no" nameDB="DESCONTO" name="desconto"
         length=”8.2” >
    <literal type="float"/>
  </field>
</struct>
<class nameDB="T_PESSOA" name="Pessoa" idt="pessoa">
  <attribute type="literal" nullable="no" nameDB="NOME" name="nome" length=”80”>
    <literal type="string"/>
  </attribute>
  <attribute type="literal" nullable="no" nameDB="CPF" name="cpf" length=”11”>
    <literal type="string"/>
  </attribute>
  <attribute type="literal" nullable="no" nameDB="DAT_NASC" name="dat_nasc">
    <literal type="date"/>
  </attribute>
</class>
```

Figura 6.7: Definição, em XML-ODMG, dos tipos estruturados `Endereco` e `Salario` e da classe `Pessoa` do esquema de origem

Os tipos estruturados do banco de dados, devem ser definidos como um `struct` da XML-ODMG, no caso dos tipos `T_ENDERECO` e `T_SALARIO`, ou como um classe, sem `extent`, como no caso do tipo `T_PESSOA`. Em ambos os casos podem ser definidos atributos de um destes tipos, como mostra a definição, em XML-ODMG, da entidade `Empregado` na figura 6.8.

```
<class nameDB="T_EMPREGADO" name="Empregado" idt="empregado"
       extent="EMPREGADOS">
  <attribute type="literal" nullable="no" nameDB="CODIGO" name="codigo" key="yes"
             length=”4”>
    <literal type="short"/>
  </attribute>
  <attribute type="class" nullable="no" nameDB="PESSOA" name="pessoa">
    <objectType ref="pessoa"/>
  </attribute>
  <attribute type="struct" nullable="no" nameDB="ENDERECO" name="endereco">
    <objectType ref="endereco"/>
  </attribute>
  <attribute type="struct" nullable="no" nameDB="SALARIO" name="salario">
    <objectType ref="salario"/>
  </attribute>
  <attribute type="literal" nullable="no" nameDB="DATAS_SALARIO"
             name="data_salario" collectionType="set" collection="yes">
    <literal type="date"/>
  </attribute>
  <attribute type="relationship" nullable="no" nameDB="CARGO" name="cargo">
    <relationship relationshipRef="e_exercido" name="exerce" classRef="cargo"/>
  </attribute>
</class>
```

Figura 6.8: Definição, em XML-ODMG, da classe `Empregado` do esquema de origem

Os atributos `pessoa`, `endereco` e `salario` fazem referência a seus respectivos tipos através do elemento `objectRef`. O *script* da figura 6.9 realiza a criação do tipo `T_EMPREGADO` e da *object-table* `EMPREGADOS`.

```
/* Criação do tipo que representa um conjunto de datas  */
create or replace type T_Datas_Salario as varray(12) of date
/

/* Criação do tipo que representa a classe Empregado */
create or replace type T_Empregado as object (
  codigo   number(8),
  pessoa   T_Pessoa,
  endereco T_Endereco,
  salario  T_Salario,
  datas_salario T_Datas_Salario,
  cargo ref T_Cargo
)
/

/* Criação da object-table que representa as instâncias da classe Empregado */
create table empregados of T_Empregado (
  codigo not null primary key
)
/
```

Figura 6.9: *Script* de criação da *object-table* `Empregados` do esquema de origem

O atributo `datas_salario` é um conjunto de datas de 12 posições, este tipo de atributo deve ser implementado, no *Oracle*, através da utilização do tipo `VARRAY` de 12 posições. Em XML-ODMG este atributo é tratado como uma coleção do tipo `set` de literais do tipo `DATE`.

A entidade `Empregado` faz um referência a entidade `Cargo` o que caracteriza um relacionamento entre classes. No banco de dados objeto-relacional este relacionamento é representado através da utilização do tipo `REF` definido no atributo de nome "cargo", o qual cria um "apontador" de uma tupla da *object-table* `EMPREGADOS` para uma outra tupla da *object-table* `CARGOS`. Em XML-ODMG este atributo é representado como um *relationship*, fazendo a referência para a classe `Cargo`. Apesar da definição deste atributo em XML-ODMG indicar o relacionamento inverso, não se faz necessária a sua implementação no banco de dados.

6.2.2 Definição do modelo de destino

O modelo de destino consiste em apenas duas entidade, `Funcionario` e `Salario`, onde a primeira é equivalente a entidade `Empregado` do modelo de origem com algumas diferenças (conflitos) que serão identificadas e resolvidas no decorrer desta seção. A figura 6.10 apresenta a definição deste modelo.

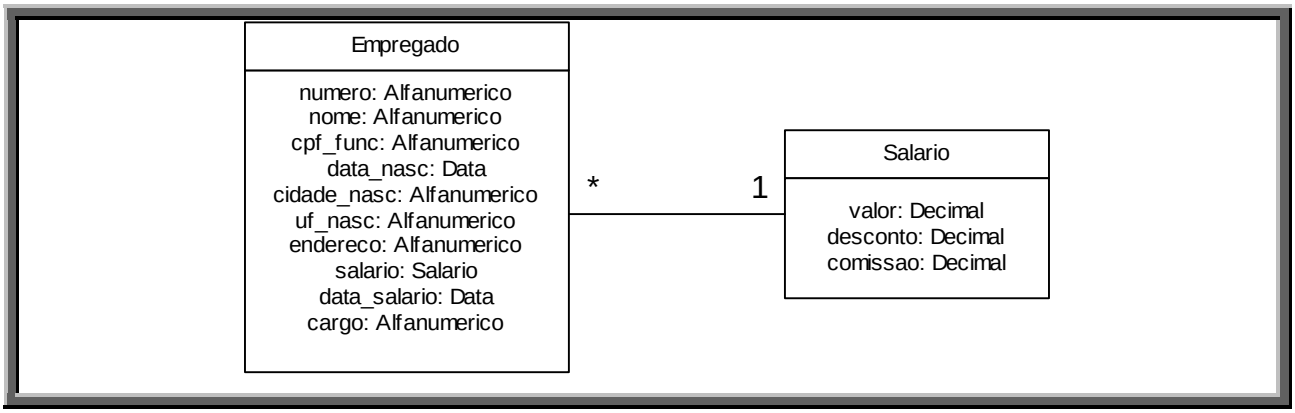

Figura 6.10: Definição conceitual das entidades do modelo de destino

```
<xml-odmg username="destino" urlJDBC="jdbc:oracle:thin:@servidor:1521:oracle"
          typeDB="OBJECT-RELATIONAL" timestampFormat="HH:mm:ss"
          dbms="oracle" password="destino" dateFormat="dd/MM/yyyy">
. . .
</xml-odmg>
```

Figura 6.11: Definição do cabçalho do XML-ODMG de Transformação e Resolução de Conflito

Como no documento do modelo de origem, o XML-ODMG de Transformação e Resolução de Conflitos necessita de informações como: nome e senha do usuário para acesso ao banco de dados de destino, *string* de conexão JDBC, qual SGBD e seu tipo e o formato de tratamento dos dados do tipo *date* e *timestamp.* A figura 6.11 mostra a definição destas informações em XML-ODMG.

A implementação deste modelo no SGBD *Oracle 8i* é idêntica a realizada no modelo de origem. Desta forma a entidade `Funcionario` vai ser implementada como tipo `T_FUNCIONARIO` no *Oracle,* o qual dará origem a *object-table* `FUNCIONARIOS`. O *script* da figura 6.12 cria os elementos do esquema da base de dados do modelo de destino.

```
/* Criação do tipo T_SALARIO */
create or replace type t_salario as object (
  valor number(10,2),
  desconto number(10,2),
  comissao number(10,2)
)
/

/* Criação do tipo T_FUNCIONARIO */
create or replace type t_funcionario as object (
  numero varchar2(10),
  nome   varchar2(80),
  cpf_func char(11),
  dat_nasc date,
  cidade_nasc varchar2(80),
  uf_nasc char(2),
  endereco varchar2(100),
  salario t_salario,
  data_salario date,
  cargo varchar2(80)
)
/

/* Criação da object-table FUNCIONARIOS */
create table funcionarios of T_Funcionario (
  numero not null primary key
)
/
```

Figura 6.12: *Script* de criação da base de dados do esquema de destino

No momento da definição do XML-ODMG de Transformação e Resolução de Conflitos deve-se identificar os conflitos existentes entre os modelos de dados envolvidos e decidir por uma solução para resolução destes.

No exemplo utilizado neste capítulo o primeiro conflito identificado é o de nomes de elementos, que aparece tanto para entidades como para atributos. A resolução para este conflito, em XML-ODMG, é apresentado na figura 6.13.

```
<class nameDB="T_FUNCIONARIO" name="Empregado" newName="Funcionario"
        idt="funcionario" extent="FUNCIONARIOS">
   <attribute type="literal" nullable="no" nameDB="NUMERO" name="codigo"
              newName="numero" key="yes" length="6">
      <literal type="string"/>
   </attribute>
  ...
</class>
```

Figura 6.13: Resolução, em XML-ODMG, do conflito de nomes da aplicação-exemplo

A classe `Funcionario`, é equivalente a classe `Empregado` do modelo de origem, desta forma deve-se utilizar o atributo `newName`, com o valor “Funcionario”. Os atributos do

XML-ODMG `nameDB` e `extent` fazem referência aos elementos do banco de dados de destino, como o tipo, `T_FUNCIONARIOS` e a *object-table* `FUNCIONARIOS`. O mesmo deve ser feito para resolver o conflito de nomes entre os atributos `codigo`, da classe `Empregado`, e `numero` da classe `Funcionario`, onde o valor do atributo `newName` deve conter o valor "numero" indicando a equivalência com o atributo `codigo`..

O segundo conflito observado é referente à estrutura do atributo. O atributo `endereco` da classe `Empregado` é composto por uma estrutura de atributos, enquanto seu equivalente na classe `Funcionario`, do modelo de destino, é do tipo *string*. Para resolver este conflito, foi decidido concatenar os campos que formam o endereço da classe `Empregado`, de maneira a se obter um valor único para preenchimento do atributo `endereco` da classe `Funcionario`. A figura 6.14 apresenta a resolução deste conflito em XML-ODMG.

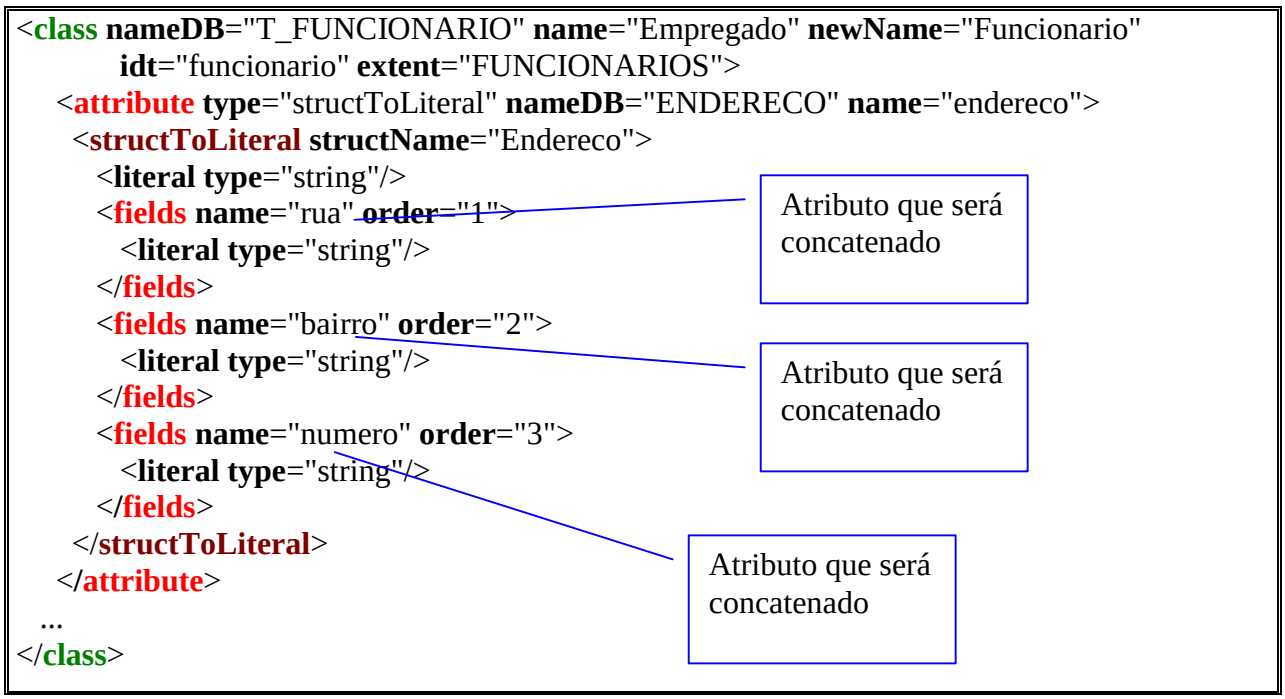

Figura 6.14: Resolução, em XML-ODMG, do conflito de composição do atributo da aplicação-exemplo

Através da definição da figura 6.14, os campos `rua`, `bairro` e `numero` serão concatenados para formar o valor que será atribuído ao campo `endereco` de uma tupla da *object-table* `FUNCIONARIOS`.

Outro conflito que pode ser observado é referente ao atributo `salario` de ambas as classes. Ambos são formados por uma estrutura, porém a que compõe a entidade

`Funcionario` possui um atributo a mais, `comissao`, em relação ao atributo equivalente na entidade `Empregado`. Para resolver este conflito foi determinado o valor 0 para preencher o campo `comissao` do atributo `salario` nas tuplas importadas para a *object-table* `FUNCIONARIOS`. A definição da resolução deste conflito, em XML-ODMG, pode ser observada da figura 6.15.

```
<struct nameDB="T_SALARIO" name="Salario" idt="salario">
  <field type="literal" nameDB="VALOR" name="valor" length=”10.2”>
    <literal type="float"/>
  </field>
  <field type="literal" nameDB="DESCONTO" name="desconto" length=’10.2”>
    <literal type="float"/>
  </field>
  <newField name="comissao" type="literal" nameDB="COMISSAO" length”10.2”>
    <literal type="float"/>
    <newValue val="0"/>
  </newField>
</struct>

<class nameDB="T_FUNCIONARIO" name="Empregado" newName="Funcionario"
        idt="funcionario" extent="FUNCIONARIOS">
   ...
   <attribute type="struct" nameDB="SALARIO" name="salario">
     <objectType ref="salario"/>
   </attribute>
</class>
```

Valor para preenchimento

Definição do atributo salário

Figura 6.15: Resolução, em XML-ODMG, do conflito de quantidades diferentes de atributos da aplicação-exemplo

Os atributos equivalentes `datas_salario` e `data_salario` do modelo de origem e destino, respectivamente, manifestam um conflito na estrutura de seu tipo, pois o atributo `data_salario` está definido como um conjunto de valores do tipo data e seu equivalente pelo tipo *date*. Utilizando a XML-ODMG, este conflito é resolvido escolhendo um determinado valor dentro do conjunto, que será utilizado para preencher o atributo no modelo de destino como mostra a figura 6.16.

```
<class nameDB="T_FUNCIONARIO" name="Empregado" newName="Funcionario"
        idt="funcionario" extent="FUNCIONARIOS">
   ...
   <attribute type="collectionToLiteral" nameDB="DATA_SALARIO" name="data_salario">
     <collectionToLiteral idx="1">
       <literal type="date"/>
     </collectionToLiteral>
   </attribute>
</class>
```

Figura 6.16: Resolução, em XML-ODMG, do conflito *coleção para literal* da aplicação exemplo

Pela definição XML-ODMG da figura 6.16, o item de índice 1 da coleção do atributo do modelo de destino deve preencher o seu atributo equivalente no modelo de destino. Assim, quando uma tupla for inserida no banco de dados de destino o atributo `DATA_SALARIO` vai receber apenas o primeiro valor do atributo `DATAS_SALARIO` das tuplas exportadas do banco de dados de origem.

Outro conflito que pode ser observado entre os modelos está relacionado às entidades `Cargo` e `Empregado` do modelo de origem e ao atributo `cargo` do modelo de destino. No esquema de origem, a entidade `Empregado` faz referência a entidade `Cargo` que indica o cargo ocupado pelo empregado, o que não acontece no esquema de destino onde o cargo é definido como um atributo do tipo *string.* Para resolver este conflito utilizando XML-ODMG, deve ser escolhido um ou vários atributos da entidade Cargo que irão ser utilizados para preencher o valor do atributo `cargo` do modelo de destino. Nesta aplicação-exemplo foi escolhido o atributo `descrição.` Esta definição pode ser observada na figura 6.17.

```
<class nameDB="T_FUNCIONARIO" name="Empregado" newName="Funcionario"
        idt="funcionario" extent="FUNCIONARIOS">
   <attribute type="relationShipToLiteral" nameDB="CARGO" name="cargo">
     <structToLiteral structName="Cargo">
       <literal type="string"/>
       <fields name="descricao" order="1">
         <literal type="string"/>
       </fields>
     </structToLiteral>
   </attribute>
   ...
</class>
```

Atributo escolhido

Figura 6.17 : Resolução, em XML-ODMG, do conflito *antidade vs atributos* da aplicação-exemplo

No momento da exportação da entidade `Empregado` do banco de dados de origem, são recuperadas as tuplas da *object-table* `EMPREGADOS` como também as da *object-table* `CARGOS` que foram referenciadas. Assim no momento da importação, quando uma tupla da *object-table* `EMPREGADOS` está sendo inserida na *object-table* `FUNCIONARIOS,` o valor do campo `cargo` é preenchido com o valor do campo `descricao` da tupla da *object-table* `CARGOS` que foi referenciada.

6.3 Conclusão

Podemos concluir que a utilização do SIBD XML-ODBMS se resume, no essencial, à definição de documentos XML-ODMG para a descrição do esquema e dos dados do BD origem, e para a resolução dos conflitos que possam existir entre os banco de dados origem e destino. As operações propriamente ditas de importação/exportação de dados são por natureza muito simples, não exigindo nenhuma inovação tecnológica.

Ainda podemos verificar, que a utilização dos documentos em XML-ODMG oferece grande flexibilidade, devido ao grande nível de detalhamento do esquema de dados permitindo que a estrutura do banco de dados possa ser fielmente representada e por permitir que sejam definidas regras para a resolução dos conflitos, não inviabilizando a operação de intercâmbio de dados.

Capítulo 7

Conclusões e Perspectivas

Sistemas de Intercâmbio de Dados entre Bancos de Dados (SIBD) são aplicações que estão ganhando importância no cenário atual. Isto porque existe hoje uma grande necessidade das organizações em integrar suas aplicações a fim de possibilitar o compartilhamento de informações de forma a trazer benefícios para o seu crescimento. Esta integração pertence à família de aplicações chamada de EAI ("**E**nterprise **A**pplication **I**ntegration").

Geralmente, o compartilhamento de informações é precedido por transações de intercâmbio de dados entre os bancos de dados das Organizações. É neste momento que os SIBDs assumem um papel importante no processo de integração de sistemas. Para que um SIBD possa ser utilizado com sucesso, é necessário que ele possa operar com diferentes plataformas como *Windows, Linux, Solaris,* etc, como também com diferentes Sistemas de Gerência de Banco de Dados (SGBD) como *Oracle, Informix, Interbase, DB2* entre outros. Essa situação de muita heterogeneidade é provavelmete típica do universo das organizações.

Para atender aos requisitos de heterogeneidade, a maioria dos SIBDs foram concebidos com as tecnologias Java e XML, pois ambas oferecem a transportabilidade necessária ao funcionamento de SIBDs. Assim alguns produtos baseados nessas tecnologias surgiram, como *XML-DBMS, DB2XML, XSU, XLE* entre outros.

A principal limitação encontrada, mediante um trabalho de análise individual de cada uma destas ferramentas feito no capítulo 3, consiste no não atendimento à banco de dados orientados a objetos. Além desta limitação, verifica-se uma carência quanto ao suporte à banco de dados objetos-relacionais, uma vez que dentre as tecnologias analisada apenas uma, XSU, permite a utilização de banco de dados objeto-relacionais, sendo este restrito apenas a SGBDs *Oracle.*

Diante destas limitações, a proposta deste trabalho destinou-se ao desenvolvimento de um SIBD que fosse capaz de trabalhar com bancos de dados orientados a objetos e objeto relacionais. Com este propósito, construimos o SIBD XML-ODBMS, um SIBD capaz de realizar o intercâmbio de dados entre banco de dados orientado a objeto e objeto-relacional, não excluindo os bancos puramente relacionais, utilizando uma linguagem de definição de dados em XML, chamada de XML-ODMG, que também foi concebida neste trabalho. A

XML-ODMG é uma linguagem de representação de dados baseada em XML definida sobre o padrão ODMG.

Alguns aspectos do XML-ODBMS podem ser destacados como pontos de diferenciação em relação a outras ferramentas da mesma família. São eles:

- Suporte a banco de dados orientados-objetos e objeto-relacionais, potencial ainda inexplorado por outros SIBDs;
- Utilização de uma linguagem própria de representação de dados em XML, XML-ODMG, definida sobre o padrão ODMG, que garante ao usuário facilidade no entendimento das informações, uma vez que as informações são representadas conforme um padrão, e flexibilidade na representação dos dados, permitindo que qualquer modelo de dados possa ser definido em XML-ODMG;
- Suporte a resolução de conflitos existentes entre as bases de dados envolvidas na operação de intercâmbio. Resolver conflitos consiste em definir regras que devem ser seguidas sempre que defirenças forem encontradas entre os modelos de dados envolvidos. A linguagem XML-ODMG permite ao usuário definir suas próprias regras as quais vão ser interpretadas e executadas pelo XML-ODBMS. A resolução de conflitos é uma funcionalidade exclusiva do XML-ODBMS em relação aos SIBDs analisados neste trabalho;
- O XML-ODBMS permite que diferentes SGBDs possam ser suportados, através dos *drivers* XML-ODBMS, sem a necessidade realizar manutenção no código fonte da ferramenta. Esta flexibilidade garante ao XML-ODBMS suporte a qualquer SGBD, sem a limitação imposta por outras ferramentas que oferecem suporte apenas a SGBDs previamente definidos no seu desenvolvimento.

XML-ODBMS é uma ferramenta inovadora dentro do universo dos SIBDs, devido aos aspectos que viemos a enumerar, com destaque para o que diz respeito ao suporte a banco de dados orientados a objetos e objeto-relacionais, flexibilidade para operar sobre diversos SGBDs e na resolução de conflitos.

No que tange às perspectivas, temos que considerar a incorporação de novas funcionalidades ao XML-ODBMS, de forma a torná-lo capaz de operar sobre diversas situações de operações de intercâmbio de dados.

O XML-ODBMS realiza o intercâmbio de dados através da exportação dos dados de uma base de dados, e na importação destes para uma outra base de dados. A operação de

importação consiste apenas no processo de inclusão, porém pode ser necessário realizar também, alterações e exclusões conforme informações oriundas da base de origem. Por exemplo, o XML-ODBMS pode ser utilizado para realizar a replicação de dados de forma a sincronizar duas bases de dados, desta forma é de fundamental importância que sejam realizadas operações de inclusão, alteração e exclusão.

Outra funcionalidade que pode ser incorporada está relacionada à definição das regras para a resolução de conflitos na XML-ODMG. A XML-ODMG suporta a resolução para uma série de conflitos apresentados no capítulo 4, mas ainda podem ser observadas diferenças não suportadas que podem inviabilizar a operação de intercâmbio de dados. Desta forma, pode-se incluir suporte a estes conflitos diminuindo a possibilidade de insucesso nas operações de intercâmbio.

Ao analisarmos os SIBDs existentes verificamos que alguns oferecem suporte ao tratamento dos dados exportados para exibição, funcionalidade que pode ser incorporada ao XML-ODBMS. Por exemplo, um documento XML-ODMG de Dados pode ser interpretado de forma a orgarnizar seus dados para serem exibidos em uma página Web, uma opção que pode ser bastente empregada em *sites* de comércio eletrônico.

O XML-ODBMS oferece suporte basicamente a três tipos de dados: numérico, texto e data/hora. Os SGBDs, geralmente, oferecem suporte a tipos de dados como imagem, som, vídeo entre outros, os quais não são suportados pelo XML-ODBMS. Definir uma forma de mapear estes tipos de dados para documentos XML-ODMG a fim de possibilitar que o XML-ODBMS possa operá-los é um trabalho de pesquisa necessário, a fim de oferecer maior flexibilidade à ferramenta.

Em relação à construção dos documentos de definição, como XML-ODMG de Esquema e de Transformação e Resolução de Conflitos, o XML-ODBMS não oferece nenhuma assistência quanto a construção destes documentos. Desta forma é interessante que sejam incorporados mecanismos que ajudem o usuário a construir estes documentos assistidos por processos automatizados de forma a facilitar o trabalho de definição de seus elementos.

Pelo fato de a arquitetura do XML-ODBMS ser baseada em componentes, a incorporação destes novos recursos pode ser realizada sem a necessidade de realizar grandes alterações no núcleo do XML-ODBMS.

Bibliografia

[ABI 00] ABI. S. “Data on the Web”. Morgan Kaufmann Publishers. Janeiro 2000

[ARM 01] ARMSTRONG, E. “Working with XML. Tha Java API for XML (JAXP) Tutorial”Agosto 2001

[BIE 00] BIERMAN, G. M. "Using XML as an Object Interchange Format". May 2000.

[BOS 97] BOS, B. “XML Representation of a relation database”. Junho 1997. Extraído em 18 de janeiro de 2001 do WWW: http://www.w3.org/XML/RDB.html

[BOS 97] BOSAK J. "XML, Java, and Future of the Web". Março 1997. Extraído em 23 de outubro de 2001 do WWW: http://citeseer.nj.nec.com/bosak97xml.html

[BOU 00] BOURRET, R. “XML Database Products”. Nov 2000, p.1-27.Extraído em 23 de fevereiro de 2001 do WWW: http://www.rpbourret.com/xml/XMLDatabaseProds.html

[BOU2 00] BOURRET, R. “XML and Databases”. Nov 2000, p.1-11.Extraído em 23 de fevereiro de 2001 do WWW:http://www.rpbourret.com/xml/XML and Databases.html

[BOU 01] BOURRET, R. “Mapping DTDs to Database” Mai 2001. Extraído em 14 de maio de 2001 do WWW: www.xml.com/lpt/a/2001/05/09/dtdtodbs.html

[CAT 00] R.G.G. CATTELL et al. "The Object Data Standerd: ODMG 3.0". Morgan Kaufmann, 2000

[DAT 00] DATE, C. J. "Introdução a Sistema de Banco de Dados" Rio de Janeiro: Editora Campus 2000

[FIS 00] FISHER, M. “Using SQL 3 Datatypes”. 2000. Extraído em 27 de agosto de 2001 do WWW: http://java.sun.com/docs/books/tutorial/jdbc/index.html

[FUH 99] Y. Fuh, S. Dessloch, W. Chen, N. Mattos, B. Tran, B. Linsday, L. DeMichiel, S. Rielau, D. Mannhaupt “Inplementation of SQL3 Structered Types with Inheritance and Value Substitutabilty” 25th VLDB Conference, Edinburgh, Scotland, 1999

[GRA 02] GRANDINETTI, L. “EAI – Integração de Aplicações Corporativas” Developers Magazine. Julho 200, p.24-30.

[ITW 02] “As iniciativas de integração geram ganhos atrativos”. IT Web. Julhho 2002. Extraído do WWW: www.itweb.com.br/noticias/asp?id=26155

[KIM 93] Kim W et al. “On resolving Schematic Heterogeneity in Multi-database Systems, in: Distributed and Parallel Databases: An International Journal”, Vol. 1, No. 3, July 93, pp. 527-549, 1993.

[LAD 00] LADDAD, R. "XML API for Database". Fev 2000. Extraído em 22/04/2001 do WWW: http://developer.java.sun.com/developer/technicalArticles/xml/api/

[LIV 00] I. Monalescu, D. Florescu, D. Kossmann, F. Xhumari, D. Olteanu “Agora: Living with XML and Relational” 26th VLDB Conference, Cairo, Egypt, 2000

[LOP 01] W. May “LoPIX: A Sytem for XML Data Integration and Manipulation” 27th VLDB Conference, Roma, Italy, 2001

[MAC 99] MACGRATH, SEAN, XML Aplicações Práticas. Como desenvolver aplicações de comércio eletrônico. Rio de Janeiro: Editora Campus, 1999.

[MCC 97] MCCLURE, S. “Object Database vs Object-Relational Database IDC Bulletin #14821E”. Agosto 1997. Extraído em 01 de setembro de 2001 do WWW: http://www.cai.com/products/jasmine/analyst/idc/14821Eat.htm

[MOR 00] MORGENTHAL, JP. "Portable Data/Portable Code: XML & Java Technologies", Mai 2000. Extraído em 15/05/2001 do WWW: http://www.javasoft.com/xml/ncfocus.html

[ORA 99] “Oracle 8i JDBC Devoloper’s Guide and Reference”. Fevereiro 1999

[ORA 00] "Oracle 8i JDBC Developer's Guide and Reference Release 3 (8.1.7)". Julho de 2000

[QUE 01] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk “Querying XML Views of Relation Data” 27th VLDB Conference, Roma, Italy, 2001

[REN 02] MESQUITA R. “Integrar é Agregar”. Informationweek. Mai 2002. Extraído em 05 de maio de 2002 do WWW: http://www.informationweek.com.br/techreport/artigo.asp?id=2387

[SAL 97] SALTOR F. et ali."On Intelligent Access to Heterogeneous Information" 1997

[SCH 01] Schloss, Bob. "Ten best bets for XML applications" Abr 2001. Extraído em 06 de março de 2002 do WWW: http://www-106.ibm.com/developerworks/xml/library/tenxmlapps/index.html?dwzone=xml

[SEL 01] SELIGMAN, L. et al “XML’s Impact on Database and Data Sharing”. Junho 2001

[SHA 00] J. Shanmugasun, E. Shekita, R. Barr, M. Carey, B. Linsday, H. Pirahesh, B.Reinwald “Efficintely Publishing Relation Data as XML Documents” 29th Conference on Very Large Database, Cario, Egypt, 2000

[SUN 99] “Sun Microsystem JDBC 2.1 API” 1999

[SUN1 00] “Building Business-to-Business Applications". Sun Microsystems. Mar 2001. Extraído em 19 de abril de 2001 do WWW:http://java.sun.com/xml/b2b.html

[SUN2 01] “Web Services Made Easier. The Java API for XML. A Techinical White Paper”.Sun Microsystems. Junho 2001

[TUR 00] TURAU, V. “Making legacy data accessbile for XML aplicationns”

[TUR 01] TURAU, V. “DB2XML, Transforming relational databases into XML documents.” Out 2001 Extraído em 28 de fevereiro de 2001 do WWW: http://www.informatik.fh- wiesbaden.de/~turau/DB2XML/index.html

[UEY 00] UEYAME, J. “A aplicação de catálogos XML no comércio eletrônico” Developers Magazine. Julho 2000, p.20-28.

[XLE 99] “XML Lightweight Extractor XLE” Extraído em 13 de maio de 2001 do WWW: http:// www.alphaworks.ibm.com/tech/xle

[XML 02] “Extensible Markup Language (XML)” Nov 2002. Extraído em 01/12/2002 do WWW: http://www.w3.org/XML/

[XML 00] “XML-DBMS, Version 1.01 Java Packages for Transferring Data between XML Documents and Relational Databases". Extraído em 28 de fevereiro de 2001 do WWW: http://www.rpbourret.com/xmldbms/index.htm

[XPE 00] M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, S. Subramanian “XPERANTO: A Middleware for Publishing Object-Relational Data as XML Documents”

[XQL 98] ROBIE, J. LAPP, J. SCHACH, D. XML Query Language (XQL). In: W3C WORKSHOP ON QUERY LANGUAGES (QL'98), 1998, Boston. Extraído em 28 de setembro de 2002 do WWW: http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[XSU 01] “XML-SQL Utility (XSU)” Extraído em 21 de setembro de 2001 do WWW: http://technet.oracle.com/docs/tech/xml/oracle_xsu/doc_library/adx04xsu.html

[WEB 01] J. Pereira, F. Fabret, H. Arno Jacobsen, F. Llirbat “WebFilter: A High-throughput XML-based Publish and Subscribe System” 27th VLDB Conference, Roma, Italy, 2001

[WEK 01] “As tecnologias que marcaram 2000”. InformationWeek,Jan 2001, p.1-4.Extraído em 20 de fevereiro de 2001 do WWW:http://www.informationweek.com.br/templates/print_story.as?id=10017

[WEK2 01] "Super bonde do comércio eletrônico",InformationWeek. Abr 2001. Extraído em 18/04/2001 do WWW: http://www.informationweek.com.br/noticias/artigo.asp?id=11584

[WIL 00] WILLIAMS, K. Professional XML Datbases. Canadá. Ed. Wrox, 2000

Apêndice A

DTDs dos XML-ODMGs de Esquema e de Transformação e Resolução de Conflitos

A.1 DTD XML-ODMG de Esquema

```
<!ELEMENT xml-odmg (struct|class|interface)* >
  <!ATTLIST xml-odmg typeDB (OBJECT-RELATIONAL|ORIENTED-OBJECT|
                             RELATIONAL) #REQUIRED
                     urlJDBC CDATA #REQUIRED
                     username CDATA #REQUIRED
                     password CDATA #REQUIRED
                     dbms (oracle|postgreesql) #REQUIRED
                     dateFormat CDATA #REQUIRED
                     timestampFormat CDATA #REQUIRED >

<!ELEMENT enum (values+)>
<!ELEMENT values (#PCDATA)>

<!ELEMENT literal (enum?) >
 <!ATTLIST literal type (long|long_long|short|unsigned_long|
                         unsigned_short|float|double|boolean|
                         octet|char|string|time|date|timestamp|enum)
"string" >

<!ELEMENT objectType EMPTY>
 <!ATTLIST objectType ref IDREF #REQUIRED>

<!ELEMENT field (literal | relationship | objectType) >
 <!ATTLIST field
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED
    nullable (yes | no) "no"
    collection (yes | no) "no"
    collectionType (set|bag|list|array|dictionary|null) "null"
    type (literal|relationship|struct|class) "literal"
    length CDATA "null">

<!ELEMENT struct
   (field)+ >
 <!ATTLIST struct
    idt ID #REQUIRED
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED  >

<!ELEMENT relationship EMPTY>
 <!ATTLIST relationship
    name CDATA #REQUIRED
    classRef IDREF #REQUIRED
    relationshipRef CDATA #REQUIRED >
```

```
<!ELEMENT attribute (literal | relationship | objectType ) >
 <!ATTLIST attribute
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED
    key (yes|no) "no"
    nullable (yes | no) "no"
    collection (yes | no) "no"
    collectionType (set|bag|list|array|dictionary|null) "null"
    type (literal|relationship|struct|class) "literal"
    length CDATA "null">

<!ELEMENT class (attribute+, objectType?) >
 <!ATTLIST class
    idt  ID #REQUIRED
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED
    extent CDATA "null">

<!ELEMENT interface (attribute+)  >
 <!ATTLIST interface
    idt  ID #REQUIRED
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED >
```

A.2 DTD XML-ODMG de Transformação e Resolução de Conflitos

```
<!ELEMENT xml-odmg (struct|class|interface)* >
  <!ATTLIST xml-odmg typeDB (OBJECT-RELATIONAL|ORIENTED-OBJECT|
                             RELATIONAL) #REQUIRED
                     urlJDBC CDATA #REQUIRED
                     username CDATA #REQUIRED
                     password CDATA #REQUIRED
                     sgbd (oracle|postgreesql) #REQUIRED
                     dateFormat CDATA #REQUIRED
                     timestampFormat CDATA #REQUIRED >
<!ELEMENT enum (values+)>

<!ELEMENT values (#PCDATA)>

<!ELEMENT literal (enum?) >
  <!ATTLIST literal type (long|long_long|short|unsigned_long|
                          unsigned_shor|float|double|boolean|
                          octet|char|string|time|date|timestamp|enum)
                          "enum" >
<!ELEMENT objectType EMPTY>
  <!ATTLIST objectType ref IDREF #REQUIRED>

<!ELEMENT fields (literal)>
  <!ATTLIST fields
      order CDATA "1"
      caracter CDATA " "
      name CDATA #REQUIRED
      length CDATA “null”>

<!ELEMENT structToLiteral (literal,fields+)>
  <!ATTLIST structToLiteral
      structName CDATA #REQUIRED >

<!ELEMENT collectionToLiteral (literal)>
  <!ATTLIST collectionToLiteral
      idx CDATA #REQUIRED>

<!ELEMENT field (literal|relationship|objectType|structToLiteral|
                 collectionToLiteral) >
 <!ATTLIST field
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED
    newName CDATA "null"
    nullable (yes | no) "no"
    collection (yes|no) "no"
    collectionType (set|bag|list|array|dictionary|null) "null"
    type (literal|relationship|struct|class|
structToLiteral|collectionToLiteral|literalToCollection|relationShipToLiter
al) "literal">

<!ELEMENT struct (field+,newField*) >
  <!ATTLIST struct
     idt ID #REQUIRED
     name CDATA #REQUIRED
     nameDB CDATA #REQUIRED
     newName CDATA "null" >

<!ELEMENT relationship EMPTY>
```

```
 <!ATTLIST relationship
    name CDATA #REQUIRED
    classRef IDREF #REQUIRED
    relationshipRef CDATA #REQUIRED >

<!ELEMENT attribute (literal|relationship|objectType|structToLiteral|
                     collectionToLiteral) >
  <!ATTLIST attribute
     name CDATA #REQUIRED
     key (yes | no) "no"
     nameDB CDATA #REQUIRED
     newName CDATA "null"
     nullable (yes | no) "no"
     collection (yes | no) "no"
     collectionType (set|bag|list|array|dictionary|null) "null"
     type (literal|relationship|struct|class|
structToLiteral|collectionToLiteral|literalToCollection|relationShipToLiter
al) "literal">

<!ELEMENT newValue EMPTY >
  <!ATTLIST newValue
     val CDATA #REQUIRED >

<!ELEMENT newField ((literal,newValue)|newStruct) >
  <!ATTLIST newField
     name CDATA #REQUIRED
     nameDB CDATA #REQUIRED
     type (literal|struct) "literal"
     length CDATA #REQUIRED “null”>

<!ELEMENT newStruct (newField+) >
  <!ATTLIST newStruct
     name CDATA #REQUIRED
     nameDB CDATA #REQUIRED >

<!ELEMENT newAttribute ((literal,newValue)|newStruct) >
  <!ATTLIST newAttribute
     name CDATA #REQUIRED
     nameDB CDATA #REQUIRED
     type (literal|struct) "literal"
     length CDATA “null”>

<!ELEMENT class (attribute+, objectType?, newAttribute*) >
  <!ATTLIST class
     idt  ID #REQUIRED
     name CDATA #REQUIRED
     nameDB CDATA #REQUIRED
     extent CDATA "null"
     newName CDATA "null" >

<!ELEMENT interface (attribute+, newAttribute*)  >
 <!ATTLIST interface
    idt  ID #REQUIRED
    name CDATA #REQUIRED
    nameDB CDATA #REQUIRED
    newName CDATA "null" >
```

Apêndice B

Documentos *XML-ODMG* de Esquema e Transformação e Resolução de conflitos utilizados nos modelos de dados do capítulo 4.

B.1 XML-ODMG de Esquema

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml-odmg SYSTEM "xml-odmg-odl.dtd">
<xml-odmg username="origem" urlJDBC="jdbc:oracle:thin:@host:1521:oracle"
          typeDB="OBJECT-RELATIONAL" timestampFormat="HH:mm:ss"
dbms="oracle"
          password="origem" dateFormat="dd/MM/yyyy">
   <struct name="Matricula" idt="matricula" nameDB="T_MATRICULA">
      <field name="nota" nameDB="NOTA" type="literal" length="4,2">
        <literal type="float"/>
      </field>
      <field name="faltas" nameDB="FALTAS" type="literal" length="3">
        <literal type="short"/>
      </field>
      <field name="faz" nameDB="FAZ" type="relationship" collection="yes"
             collectionType="set">
        <relationship relationshipRef="e_feita_por" name="faz"
                      classRef="disciplina"/>
      </field>
   </struct>
   <struct name="Endereco" nameDB="T_ENDERECO" idt="endereco">
      <field name="numero" nameDB="NUMERO" type="literal" length="5">
        <literal type="string"/>
      </field>
     <field name="rua" nameDB="RUA" type="literal" length="80">
        <literal type="string"/>
      </field>
     <field name="bairro" nameDB="BAIRRO" type="literal" length="80"
            nullable="yes">
        <literal type="string"/>
      </field>
     <field name="cidade" nameDB="CIDADE" type="literal" length="80">
        <literal type="string"/>
      </field>
     <field name="estado" nameDB="ESTADO" type="literal" length="2">
        <literal type="char"/>
      </field>
   </struct>
   <struct name="Desconto" nameDB="T_DESCONTO" idt="desconto">
      <field name="tipo_desconto" nameDB="TIPO_DESCONTO" type="literal"
             length="1" nullable="yes">
        <literal type="char"/>
      </field>
```

```
      <field name="valor" nameDB="VALOR" type="literal" length="10,2"
             nullable="yes">
        <literal type="float"/>
      </field>
   </struct>
   <struct name="Titulacao" nameDB="T_TITULACAO" idt="titulacao">
      <field name="titulo" nameDB="TITULO" type="literal" length="80">
        <literal type="string"/>
      </field>
      <field name="instituicao" nameDB="INSTITUICAO" type="literal"
             length="80">
        <literal type="string"/>
      </field>
      <field name="inicio" nameDB="INICIO" type="literal">
        <literal type="date"/>
      </field>
      <field name="fim" nameDB="FIM" type="literal" nullable="yes">
        <literal type="date"/>
      </field>
   </struct>
   <class name="Curso" idt="curso" nameDB="T_CURSO" extent="CURSOS">
      <attribute name="cod_curso" nameDB="COD_CURSO" key="yes"
type="literal"
                 length="4">
        <literal type="short"/>
      </attribute>
      <attribute name="nom_curso" nameDB="NOM_CURSO" type="literal"
                 length="80">
        <literal type="string"/>
      </attribute>
      <attribute name="carga_horaria" nameDB="CARGA_HORARIA" type="literal"
                 length="4">
        <literal type="short"/>
      </attribute>
      <attribute name="datas_reconhecimento" nameDB="DATAS_RECONHECIMENTO"
                 type="literal" collection="yes" collectionType="set">
        <literal type="date"/>
      </attribute>
      <attribute name="e_cursado_por" nameDB="E_CURSADO_POR"
                 type="relationship" collection="yes" collectionType="set"
>
        <relationship relationshipRef="cursa" name="e_cursado_por"
                      classRef="aluno"/>
      </attribute>
   </class>
   <class name="Pessoa" nameDB="T_PESSOA" idt="pessoa">
      <attribute name="nome" nameDB="NOME" type="literal" length="80">
        <literal type="string"/>
      </attribute>
      <attribute name="cpf" nameDB="CPF" type="literal" length="11">
        <literal type="char"/>
      </attribute>
      <attribute name="data_nasc" nameDB="DATA_NASC" type="literal">
        <literal type="date"/>
      </attribute>
      <attribute name="endereco" nameDB="ENDERECO" type="struct">
        <objectType ref="endereco"/>
      </attribute>
      <attribute name="telefones" nameDB="TELEFONES" type="literal"
                 length="20" collection="yes" collectionType="set">
        <literal type="string"/>
```

```
   </attribute>
</class>
<interface name="Aluno-IF" nameDB="T_ALUNO-IF" idt="aluno-if">
   <attribute name="matricula" nameDB="MATRICULA" type="literal"
              length="10">
      <literal type="short"/>
   </attribute>
   <attribute name="media" nameDB="MEDIA" type="literal" length="8,2">
      <literal type="float"/>
   </attribute>
   <attribute name="desconto" nameDB="DESCONTO" type="struct">
      <objectType ref="desconto"/>
   </attribute>
   <attribute name="matriculas" nameDB="MATRICULAS" type="struct"
              collection="yes" collectionType="set">
      <objectType ref="matricula"/>
   </attribute>
   <attribute name="cursa" nameDB="CURSA" type="relationship">
      <relationship relationshipRef="e_cursada_por" name="cursa"
                    classRef="curso"/>
   </attribute>
</interface>
<class name="Aluno" nameDB="T_ALUNOS" idt="aluno" extent="ALUNOS">
   <objectType ref="aluno-if"/>
   <attribute name="repreensao" nameDB="REPREENSAO" type="literal"
              length="1">
      <literal type="char"/>
   </attribute>
</class>
<class name="Centro" idt="centro" nameDB="T_CENTRO" extent="CENTROS">
   <attribute name="cod_centro" nameDB="COD_CENTRO" type="literal"
              key="yes" length="4">
      <literal type="short"/>
   </attribute>
   <attribute name="nom_centro" nameDB="NOM_CENTRO" type="literal"
              length="80">
      <literal type="short"/>
   </attribute>
   <attribute name="possui" nameDB="POSSUI" type="relationship">
      <relationship name="possui" ralationshipRef="faz_parte"
                    classRef="professor"/>
   </attribute>
</class>
<class name="Professor" idt="professor" nameDB="T_PROFESSOR"
       extent="PROFESSORES">
   <objectType ref="pessoa"/>
   <attribute name="cod_prof" nameDB="COD_PROF" type="literal" key="yes"
              length="4">
      <literal type="short"/>
   </attribute>
   <attribute name="titulacoes" nameDB="TITULACOES" type="struct"
              collection="yes" collectionType="set">
      <objectType ref="titulacao"/>
   </attribute>
   <attribute name="ensina" nameDB="ENSINA" type="relationship"
              collection="yes" collectionType="set">
      <relationship name="ensina" ralationshipRef="e_ensinada_por"
                    classRef="disciplina"/>
   </attribute>
   <attribute name="faz_parte" nameDB="FAZ_PARTE" type="relationship"
              collection="yes" collectionType="set">
```

```
         <relationship name="faz_parte" ralationshipRef="possui"
                       classRef="centro"/>
      </attribute>
   </class>
   <class name="Disciplina" idt="disciplina" nameDB="T_DISCIPLINA"
          extent="DISCIPLINAS">
      <attribute name="cod_disc" nameDB="COD_DISC" key="yes" type="literal"
                 length="4">
         <literal type="short"/>
      </attribute>
      <attribute name="nom_disc" nameDB="NOM_DISC" type="literal"
length="80">
        <literal type="string"/>
      </attribute>
      <attribute name="carga_horaria" nameDB="CARGA_HORARIA" type="literal"
                 length="4">
         <literal type="short"/>
      </attribute>
      <attribute name="e_feita_por" nameDB="E_FEITA_POR"
type="relationship"
                 collection="yes" collectionType="set">
         <relationship name="e_feita_por" ralationshipRef="faz"
                       classRef="aluno"/>
      </attribute>
      <attribute name="e_ensinada_por" nameDB="E_ENSINADA_POR"
                 type="relationship" collection="yes" collectionType="set">
         <relationship name="e_ensinada_por" ralationshipRef="ensina"
                       classRef="professor"/>
      </attribute>
   </class>
</xml-odmg>
```

B.2 XML-ODMG de Transformação e Resolução de Conflitos

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml-odmg SYSTEM "xml-odmg-trans.dtd">
<xml-odmg username="destino" urlJDBC="jdbc:oracle:thin:@host:1521:oracle"
          typeDB="OBJECT-RELATIONAL" timestampFormat="HH:mm:ss"
          dbms="oracle" password="destino" dateFormat="dd/MM/yyyy">
   <struct name="Matricula" idt="matricula" nameDB="T_MATRICULA">
      <field name="nota" nameDB="NOTA" type="literal" length="4,2">
        <literal type="float"/>
      </field>
      <field name="faltas" nameDB="FALTAS" type="literal" length="3">
        <literal type="short"/>
      </field>
      <field name="faz" nameDB="FAZ" type="relationship" collection="yes"
             collectionType="set">
        <relationship relationshipRef="e_feita_por" name="faz"
                      classRef="disciplina"/>
      </field>
   </struct>
   <struct name="Desconto" nameDB="T_DESCONTO" idt="desconto">
      <field name="tipo_desconto" nameDB="TIPO_DESCONTO" type="literal"
             length="1" nullable="yes">
        <literal type="char"/>
      </field>
      <field name="valor" nameDB="VAL_DESCONTO" newName="val_desconto"
```

```
            type="literal" length="10,2" nullable="yes">
        <literal type="float"/>
      </field>
      <newField name="situacao" nameDB="SITUACAO" type="literal"
length="1">
        <literal type="char"/>
        <newValue val="A"/>
      </newField>
   </struct>
   <struct name="Titulacao" nameDB="T_TITULACAO" idt="titulacao">
      <field name="titulo" nameDB="TITULO" type="literal" length="80">
        <literal type="string"/>
      </field>
      <field name="instituicao" nameDB="INSTITUICAO" type="literal"
             length="80">
        <literal type="string"/>
      </field>
      <field name="inicio" nameDB="INICIO" type="literal">
        <literal type="date"/>
      </field>
      <field name="fim" nameDB="FIM" type="literal" nullable="yes">
        <literal type="date"/>
      </field>
   </struct>
   <class name="Curso" idt="curso" nameDB="T_CURSO" extent="CURSOS">
      <attribute name="cod_curso" nameDB="COD_CURSO" key="yes"
type="literal"
                 length="4">
         <literal type="short"/>
      </attribute>
      <attribute name="nom_curso" nameDB="NOM_CURSO" type="literal"
                 length="80">
         <literal type="string"/>
      </attribute>
      <attribute name="datas_reconhecimento" nameDB="DATA_RECONHECIMENTO"
                 newName="data_reconhecimento" type="collectionToLiteral">
         <collectionToLiteral idx="1">
           <literal type="date"/>
         </collectionToLiteral>
      </attribute>
      <attribute name="e_cursado_por" nameDB="E_CURSADO_POR"
                 type="relationship" collection="yes" collectionType="set"
>
         <relationship relationshipRef="cursa" name="e_cursado_por"
                       classRef="aluno"/>
      </attribute>
   </class>
   <class name="Pessoa" nameDB="T_PESSOA" idt="pessoa">
      <attribute name="nome" nameDB="NOME" type="literal" length="80">
         <literal type="string"/>
      </attribute>
      <attribute name="cpf" nameDB="CPF" type="literal" length="11">
         <literal type="char"/>
      </attribute>
      <attribute name="data_nasc" nameDB="DATA_NASC" type="literal">
         <literal type="date"/>
      </attribute>
      <attribute name="endereco" nameDB="ENDERECO" type="structToLiteral">
         <structToLiteral name="Endereco">
            <literal type="string"/>
            <fields order="1" caracter=" " name="rua" length="30">
```

```
            <literal type="string"/>
         </fields>
         <fields order="2" caracter=" " name="numero" length="10">
            <literal type="string"/>
         </fields>
         <fields order="3" caracter=" " name="bairro" length="10">
            <literal type="string"/>
         </fields>
      </structToLiteral>
   </attribute>
   <attribute name="telefones" nameDB="TELEFONES" type="literal"
              length="20" collection="yes" collectionType="set">
      <literal type="string"/>
   </attribute>
</class>
<class name="Aluno" newName="Estudante" nameDB="T_ESTUDANTE"
       idt="estudante" extent="ESTUDANTES">
   <objectType ref="pessoa"/>
   <attribute name="matricula" newName="codigo" nameDB="CODIGO"
              type="literal" length="10">
      <literal type="short"/>
   </attribute>
   <attribute name="media" newName="media_geral" nameDB="MEDIA_GERAL"
              type="literal" length="4,2">
      <literal type="float"/>
   </attribute>
   <attribute name="repreensao" newName="repreensoes"
nameDB="REPREENSOES"
              type="literal" length="1">
      <literal type="char"/>
   </attribute>
   <attribute name="desconto" nameDB="DESCONTO" type="struct" >
      <objectRef ref="desconto"/>
   </attribute>
</class>
<class name="Professor" idt="professor" nameDB="T_PROFESSOR"
       extent="PROFESSORES">
   <objectType ref="pessoa"/>
   <attribute name="cod_prof" nameDB="COD_PROF" type="literal" key="yes"
              length="4">
      <literal type="short"/>
   </attribute>
   <newAttribute name="carga_hor_disponivel"
nameDB="CARGA_HOR_DISPONIVEL"
                 type="struct">
      <newStruct name="Carga_hor_turno" nameDB="T_CARGA_HOR_TURNO">
         <newField name="turno" nameDB="TURNO" length="1"
type="literal">
            <literal type="char"/>
            <newValue val="M"/>
         </newField>
         <newField name="carga_hor" nameDB="CARGA_HOR" length="3"
                   type="struct">
            <literal type="short"/>
            <newValue val="40"/>
         </newField>
      </newStruct>
   </newAttribute>
   <attribute name="faz_parte" newName="centro" nameDB="CENTRO"
              type="relationshipToLiteral" length="50">
      <structToLiteral name="Centro">
```

```
            <fields name="nom_centro" order="1" caracter=" " length="50">
               <literal type="string"/>
            </fields>
         </structToLiteral>
      </attribute>
      <attribute name="titulacoes" nameDB="TITULACOES" type="struct"
                 collection="yes" collectionType="set">
         <objectType ref="titulacao"/>
      </attribute>
      <attribute name="ensina" nameDB="ENSINA" type="relationship"
                 collection="yes" collectionType="set">
         <relationship name="ensina" ralationshipRef="e_ensinada_por"
                       classRef="disciplina"/>
      </attribute>
   </class>
   <class name="Disciplina" idt="disciplina" nameDB="T_DISCIPLINA"
          extent="DISCIPLINAS">
      <attribute name="cod_disc" nameDB="COD_DISC" key="yes" type="literal"
                 length="4">
         <literal type="short"/>
      </attribute>
      <attribute name="nom_disc" nameDB="NOM_DISC" type="literal"
length="80">
        <literal type="string"/>
      </attribute>
      <attribute name="carga_horaria" nameDB="CARGA_HORARIA" type="literal"
                 length="4">
         <literal type="short"/>
      </attribute>
      <attribute name="e_feita_por" nameDB="E_FEITA_POR"
type="relationship"
                 collection="yes" collectionType="set">
         <relationship name="e_feita_por" ralationshipRef="faz"
                       classRef="aluno"/>
      </attribute>
      <attribute name="e_ensinada_por" nameDB="E_ENSINADA_POR"
                 type="relationship" collection="yes" collectionType="set">
         <relationship name="e_ensinada_por" ralationshipRef="ensina"
                       classRef="professor"/>
      </attribute>
   </class>
</xml-odmg>
```