

Fabiana Frata Furlan Peres

**Uma Proposta de Mapeamento do Modelo de Banco de Dados Orientado a Objetos
para o Modelo XML Schema**

**Florianópolis – SC
2005**

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

Fabiana Frata Furlan Peres

**UMA PROPOSTA DE MAPEAMENTO DO
MODELO DE BANCO DE DADOS ORIENTADO A
OBJETOS PARA O MODELO XML SCHEMA**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador:
Ronaldo dos Santos Mello

Florianópolis, agosto - 2005

UMA PROPOSTA DE MAPEAMENTO DO MODELO DE BANCO DE DADOS ORIENTADO A OBJETOS PARA O MODELO XML SCHEMA

Fabiana Frata Furlan Peres

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Raul Sidnei Wazlawich, Dr

Banca Examinadora:

Ronaldo dos Santos Mello, Dr.

Frank Augusto Siqueira, Dr.

Nina Krahe Edelweiss, Dra.

João Bosco Mangueira Sobral, Dr.

Há três coisas que nunca voltam atrás: a flecha lançada, a palavra pronunciada e a oportunidade perdida. (Provérbio chinês)

Aos meus filhos Murilo e Ricardo e a meu
esposo Frank, que são a razão da minha vida.

Agradeço a todos que de alguma forma contribuíram para a concretização deste trabalho. Em especial a Deus pela minha saúde e coragem para jamais desistir de meus objetivos. Ao meu esposo que sempre me deu imenso apoio. Ao meu Orientador por ter confiado na minha capacidade e me direcionado no caminho para o meu sucesso. À minha mãe, minha cunhada Bete e minha amiga Nilse que muitas vezes deixaram suas casas para cuidar do meu filho para que eu pudesse prosseguir nesta luta. E também aos colegas de trabalho que sempre estiveram prontos a me ajudar no que fosse necessário.

Muito Obrigada, a todos vocês!

SUMÁRIO

LISTA DE ILUSTRAÇÕES	VIII
LISTA DE TABELAS	IX
RESUMO	X
ABSTRACT	XI
1. INTRODUÇÃO	1
2. BANCO DE DADOS ORIENTADO A OBJETOS	3
2.1. MODELO DE DADOS	3
2.1.1. <i>Classes e Objetos</i>	4
2.1.2. <i>Identidade do Objeto</i>	5
2.1.3. <i>Estruturas de Objetos</i>	6
2.1.4. <i>Herança</i>	7
3. XML E XML SCHEMA	9
3.1. XML	9
3.2. XML SCHEMA	10
3.2.1. <i>Recursos da linguagem XML Schema</i>	11
4. TRABALHOS RELACIONADOS	15
4.1. PROPOSTA DE HAN ET ALL	15
4.2. PROPOSTA DE RENNER	20
4.3. PROPOSTA DE CHUNG ET AL	22
4.4. PROPOSTA DE SALIM ET AL	26
5. ABORDAGEM DE MAPEAMENTO BDOO-XML SCHEMA	28
5.1. REGRAS DE MAPEAMENTO	28
5.1.1. <i>Regra CC – Mapeamento de Classe Concreta</i>	30
5.1.2. <i>Regra CA – Mapeamento de Classe Abstrata</i>	31
5.1.3. <i>Regra TP – Mapeamento de Tipo Primitivo</i>	31
5.1.4. <i>Regra ETT – Mapeamento de Estrutura Tipo Tupla</i>	32
5.1.5. <i>Regra ETL – Mapeamento de Estrutura Tipo Lista</i>	34
5.1.6. <i>Regra ETC – Mapeamento de Estrutura Tipo Conjunto</i>	36
5.1.7. <i>Regra AR – Mapeamento de Atributo de Referência</i>	37
5.1.8. <i>Regra RH1 – Mapeamento de Relacionamento de Herança Simples</i>	38
5.1.9. <i>Regra RH2 – Mapeamento de Herança Múltipla</i>	40
5.2. ALGORITMO DE MAPEAMENTO	41
6. ESTUDO DE CASO	43
6.1. REQUISITOS DO DOMÍNIO	43
6.2. APLICAÇÃO DO ALGORITMO DE MAPEAMENTO	44

7. CONCLUSÃO.....	57
BIBLIOGRAFIA	61
ANEXO 1.....	64

LISTA DE ILUSTRAÇÕES

Figura 2-1. Representação das classes de um esquema BDOO: (a) Classes concretas, (b) Classes abstratas.	5
Figura 2-2. Exemplo de Objetos Complexos.	7
Figura 2-3. Exemplo de herança.	8
Figura 3-1. Documento XML.	10
Figura 3-3. Um exemplo de esquema em XML Schema.	12
Figura 4-1. Possíveis mapeamentos de estruturas XML para um esquema de BDOO/OR [HAN03].	16
Figura 4-2. XML Schema (a), trecho XML Schema contendo informações sobre o mapeamento (b) [HAN03].	18
Figura 4-3. Classes catalogo para armazenar informações sobre o mapeamento [HAN03].	19
Figura 4-4. Algoritmo para armazenar documentos XML em um BDOO/OR [HAN03].	19
Figura 4-5. Mapeamento de XML para objetos Java [REN01].	20
Figura 4-6. Mapeamento de tipos de dados XML para objetos Java: definição de um tipo complexo(a), definição de tipo derivado(b), classe Java gerada(c) [REN01].	21
Figura 4-7. Exemplo de DTD (a), representação em grafo da DTD(b) [CHU01].	23
Figura 4-8. Um esquema para o BDOO [CHU01].	24
Figura 4-9. Uma expressão regular(a), expressão regular relaxada da expressão regular(b) [CHU01.]	24
Figura 4-10. Algoritmo de construção de um autômato [CHU01].	25
Figura 4-11. Um autômato para um trecho da DTD [CHU01].	25
Figura 4-12. Esquema para o BDOO gerado após aplicação da segunda etapa da proposta [CHU01].	25
Figura 6-1. Esquema do BDOO da realidade Universitária.	44
Figura 6-2. Representação gráfica do XML Schema da realidade universitária.	47
Figura 6-3. Esquema XML da realidade universitária.	54
Figura 6-4. Dados do BDOO.	54
Figura 6-5. Instância XML validado pelo esquema gerado.	56

LISTA DE TABELAS

Tabela 1. Resumo dos recursos da linguagem XML Schema	12
Tabela 2. Resumo das regras de mapeamento dos construtores de XML Schema [SAL04].	27
Tabela 3. Regras de mapeamento de esquemas BDOO para esquemas XML.	29
Tabela 4. Comparação entre as correspondências estabelecidas entre conceitos de BDOO e XML Schema definidos pelos trabalhos relacionados e a abordagem proposta..	58

RESUMO

Atualmente encontramos diversos domínios de aplicação com um grande número de sistemas heterogêneos e distribuídos que realizam troca de dados entre si. Estes sistemas podem utilizar modelos de dados distintos. Neste contexto, a XML possibilita a definição de uma linguagem padrão para a representação de dados de um domínio, permitindo que estes sejam compartilhados entre os sistemas. Entretanto, para um compartilhamento efetivo destes dados, cada aplicação precisa conhecer o esquema dos documentos XML que ela exporta ou importa, e também desenvolver um mecanismo de mapeamento deste esquema XML para o seu esquema de dados local.

Assim sendo, este trabalho foca em uma parte desta problemática, propondo uma abordagem de mapeamento de esquemas de bancos de dados orientados a objetos para esquemas XML. Para tanto, são definidas regras para a realização deste mapeamento e um algoritmo que apresenta uma seqüência de passos para a aplicação destas regras. O esquema XML adotado é baseado no modelo da linguagem *XML Schema*, que é a recomendação mais recente da W3C para a definição de esquemas XML.

ABSTRACT

Nowadays there are many application domains with lots of distributed and heterogeneous systems that exchange data among them. These systems may use different data models. In this context, XML enables the definition of a default language for domain data representation, allowing these data to be shared among the systems. Nevertheless, for an effective data sharing, each application needs to know the schema of the XML documents that it imports or exports, and also needs to develop a mapping mechanism to convert this XML schema into its local data schema.

This work focuses on part of this problem, proposing a mapping approach that transforms an object-oriented database schema into an XML Schema specification. In order to accomplish this, rules are defined to provide the mapping as well as an algorithm that shows the sequence of steps for applying the rules. The adopted XML schema is based on the XML Schema language model, which is the most recent recommendation of W3C for the specification of XML schemas.

1. Introdução

O uso do paradigma OO (Orientado a Objetos) cresceu muito rapidamente desde o seu surgimento devido a um conjunto de fatores, como reusabilidade, extensibilidade e nível de abstração adotado, que favorecem o desenvolvimento de aplicações robustas [PRE03]. Como consequência, surgiram diversas tecnologias para dar suporte ao desenvolvimento destas aplicações, como a UML (*Unified Modeling Language*) [BOO99], uma linguagem padrão para modelagem de aplicações OO, BDOOs (Bancos de Dados Orientados a Objetos) [BER93] que têm por objetivo armazenar os objetos persistentes de uma aplicação.

No estado atual do desenvolvimento de sistemas, existem muitos domínios de aplicações, onde há um grande número de sistemas heterogêneos e distribuídos que precisam realizar troca de dados. Cada aplicação pode utilizar um modelo de dados distinto e executar em locais geográficos diferentes.

A XML (*Extensible Markup Language*) surgiu neste contexto com o objetivo de permitir a definição de linguagens de marcação que possam ser utilizadas na representação de dados, possibilitando o compartilhamento destes dados entre as aplicações de forma simples, uma vez que documentos XML são arquivos texto. Entretanto para o compartilhamento efetivo destes dados, cada aplicação precisa conhecer o esquema dos documentos XML que compartilha e também possuir formas de mapear estes dados para o seu modelo de dados particular. Este esquema pode ser definido através da linguagem XML Schema (*Extensible Markup Language Schema*) que é atualmente a recomendação mais completa para definição de esquemas de documentos XML [W3C04].

Assim sendo, este trabalho se atém num subconjunto desta problemática, propondo uma abordagem de mapeamento de esquemas de BDOOs para esquemas XML. BDOOs formam uma categoria de BDs não convencionais já com diversos produtos no mercado, porém existe pouca literatura tratando do mapeamento XML-BDOO .

A abordagem proposta nesta dissertação tem como objetivo geral obter uma representação no modelo XML Schema para o modelo de dados de um BDOO. Já os objetivos específicos são definições de regras para esse mapeamento, e uma seqüência

de passos que apresenta a ordem de aplicação destas regras e gera um esquema XML que representa adequadamente a estrutura de um BDOO.

A contribuição deste trabalho é o intercâmbio de informações entre as aplicações, especificamente no contexto de troca de dados entre BDOOs através de esquemas XML e também a geração de esquemas XML para a publicação na Internet, de dados armazenados em BDOOs.

Os capítulos que seguem têm por objetivo contextualizar e descrever a proposta de mapeamento. O segundo capítulo contém uma descrição do modelo de dados de um BDOO e da representação lógica adotada para esquemas de dados de BDOOs. O terceiro capítulo apresenta, de forma resumida, os conceitos relacionados com a linguagem XML Schema e também descreve a representação gráfica adotada para os seus conceitos, de forma a facilitar a compreensão das regras de mapeamento. No fim deste trabalho tem um anexo que detalha estes recursos da linguagem XML Schema. O quarto capítulo apresenta trabalhos relacionados. O quinto capítulo apresenta a abordagem de mapeamento proposta, contendo as definições das regras e um algoritmo que define os passos para aplicação das regras. O capítulo seis apresenta um estudo de caso em um contexto universitário, a fim de validar a abordagem proposta. Por fim, o sétimo capítulo apresenta as conclusões feitas sobre este trabalho e também sugere trabalhos futuros que podem ser realizados para aperfeiçoá-lo.

2. Banco de Dados Orientado a Objetos

Com o surgimento de aplicações ditas não convencionais, como sistemas de informação geográfica e sistemas de CAD/CAM, houve a necessidade de se desenvolver modelos de dados mais elaborados para bancos de dados, uma vez que estas aplicações lidam com estruturas complexas de dados e realizam operações igualmente complexas sobre seus dados.

O tradicional modelo de dados relacional se mostra limitado para atender os requisitos de dados destas aplicações, uma vez que é orientado a registros. Nesta abordagem, toda *tupla* possui a mesma estrutura e seus atributos são de tamanho fixo e tipos de dados simples (atômicos). Esta abordagem não é capaz de abstrair a representação de dados complexos [KIM90].

Um BDOO é uma tecnologia que dá suporte a aplicações desenvolvidas com o uso do paradigma OO. A motivação para o desenvolvimento destes bancos de dados surgiu da dificuldade encontrada em representar objetos persistentes e operações complexas de manipulação sobre eles.

Segundo Won King [KIM90], o modelo de dados de um BDOO considera um conjunto de conceitos para representação de objetos persistentes. Alguns destes conceitos são adotados há algum tempo por diversas linguagens de programação OO, assim como por linguagens de representação de conhecimento. Entretanto, não existe um consenso sobre estes conceitos, e conseqüentemente, não há consenso para modelos de dados de BDOOs. Mesmo assim, há um conjunto de conceitos que é considerado fundamental para definir um modelo de dados básico para BDOOs [BER93]. Estes conceitos são detalhados a seguir.

2.1. Modelo de Dados

Esta seção apresenta os principais conceitos que definem a estrutura do esquema lógico de um BDOO e sua representação gráfica baseado em [GAI97]. O entendimento destes conceitos é necessário para a definição das regras de mapeamento.

2.1.1. Classes e Objetos

A classe é o conceito mais importante de um BDOO. Ela descreve as características comuns a um conjunto de objetos através de atributos, métodos e seus relacionamentos com outras classes de objetos. Um objeto é um fato do mundo real relevante para o domínio de aplicação e que deve ser persistido no BDOO.

Segundo Bertino [BER93], os BDOOs podem ser classificados em duas categorias – aqueles que suportam o conceito de classe e aqueles que suportam o conceito de tipo. O conceito de tipo é muito similar ao de TAD (Tipo Abstrato de Dados) que modela características comuns a várias classes e que é usado somente na definição de outras classes, não podendo ser instanciado.

Existem BDOOs que suportam os dois conceitos, permitindo que uma classe seja definida com base na definição de um tipo. Assim, os métodos definidos nos tipos são implementados na classe. Neste caso, a classe define a implementação do conjunto de objetos e o tipo descreve a interface e a estrutura destes objetos [KIM90]. Alguns BDOOs suportam ainda o conceito de classe abstrata, que define atributos e métodos para um conjunto de objetos, mas não pode ser instanciada.

Outro conceito associado à classe é o conceito de objeto-classe. Um objeto-classe pode conter informações comuns a todas as instâncias das classes, sendo estas informações armazenadas no banco de dados em um local separado das instâncias. Este tratamento evita o armazenamento repetido das características da classe em todas as instâncias da mesma e tira proveito da característica de reusabilidade.

A maioria dos BDOOs restringe um objeto a ser uma instância de uma única classe, embora um objeto possa ser encarado como membro de várias classes através de herança múltipla, conforme apresentado na seção 2.1.4.

Por definição, todos os objetos que existem dentro de uma classe herdam os atributos e as operações disponíveis para manipular os atributos pertencentes a esta classe [BER99].

A cada objeto está associado um estado e um comportamento. O estado de um objeto é caracterizado pelos valores armazenados num determinado momento por ele, mais especificamente por seus atributos. Estes valores podem ser modificados ao longo da vida do objeto. O comportamento de um objeto é um conjunto de ações pré-definidas

(métodos) que o objeto pode responder quando requisitado por outros objetos. O comportamento de um objeto em um determinado momento depende do seu estado.

Convenciona-se neste trabalho que as classes de um esquema OO são representadas por um retângulo dividido em duas partes. A primeira parte contém os atributos da classe e a outra é destinada à declaração de métodos. A Figura 2-1(a) mostra um exemplo de representação de classe. Já as classes abstratas são formadas pela mesma estrutura, mas para diferenciá-las das classes concretas, elas são representadas por um retângulo pontilhado como apresenta a Figura 2-1(b).

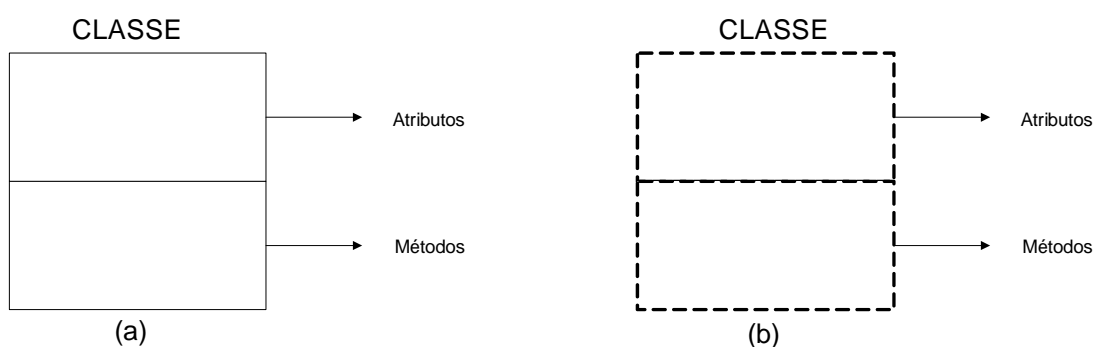


Figura 2-1. Representação das classes de um esquema BDOO: (a) Classes concretas, (b) Classes abstratas.

2.1.2. Identidade do Objeto

Cada objeto possui um identificador único chamado de *OID (Object Identifier)* pelo qual ele é identificado. Este OID, como o próprio nome já diz, é único para cada objeto e pode ser visível ou não para o usuário. Os Sistemas Gerenciadores de BDOOs (SGBDOOs) são os responsáveis por gerar e gerenciar os OIDs dos objetos, impossibilitando o usuário de fazer modificações nos mesmos. Esta característica presente nos SGBDOOs apresenta diversas vantagens, entre elas:

- não há necessidade de criar atributos adicionais e sem semântica apenas para identificar um objeto;
- o usuário não precisa realizar o gerenciamento de chaves para identificação das entidades persistentes, como ocorre no modelo relacional;

- o gerenciamento da identificação de dados apresenta uma melhor performance, visto que OIDs em geral são implementações de baixo nível do sistema.

2.1.3. Estruturas de Objetos

A cada objeto é associado um conjunto de atributos e a definição destes atributos pode gerar uma estrutura complexa para o objeto. Em um BDOO, o domínio de um atributo pode ser um tipo primitivo (inteiro, cadeia de caracteres,...), um outro objeto ou então um domínio estruturado, definido através de um chamado construtor de tipo (tupla, conjunto ou lista, em geral) que define uma estrutura complexa para o objeto. Um objeto que possui atributos cujo domínio são outros objetos ou estruturas complexas é considerado um objeto complexo [BER93]. A Figura 2-2, mostra um objeto complexo chamado *Departamento*. Ele possui o atributo *Docentes*, que é um conjunto de objetos da classe *Docente*.

Quando o domínio de um atributo é um objeto, os BDOOs geralmente armazenam neste atributo somente uma referência ao objeto através do OID deste objeto referenciado. [BER93]. Para representar a associação estabelecida entre as classes por estes atributos de referência, é utilizada na Figura 2-2 uma seta contínua. Estas referências são monovaloradas ou então multivaloradas.

Conforme descrito anteriormente, os principais construtores de tipos são conjuntos, listas e tuplas. Conjunto é o meio natural de representar coleções do mundo real e são usados para definir atributos multivalorados. Na Figura 2-2, o atributo *Docentes* do objeto *Departamento* é multivalorado (é o conjunto de *Docentes* pertencentes a um *Departamento*). Tuplas fornecem um meio de abstrair uma estrutura de registro em uma entidade, como mostra a declaração do atributo *Endereço* do objeto *Docente*. Listas são multivaloradas como os conjuntos, porém estabelecem uma ordem entre os objetos ou valores referenciados. Na Figura 2-2, o atributo *Especialidades* do objeto *Docente* é uma lista.

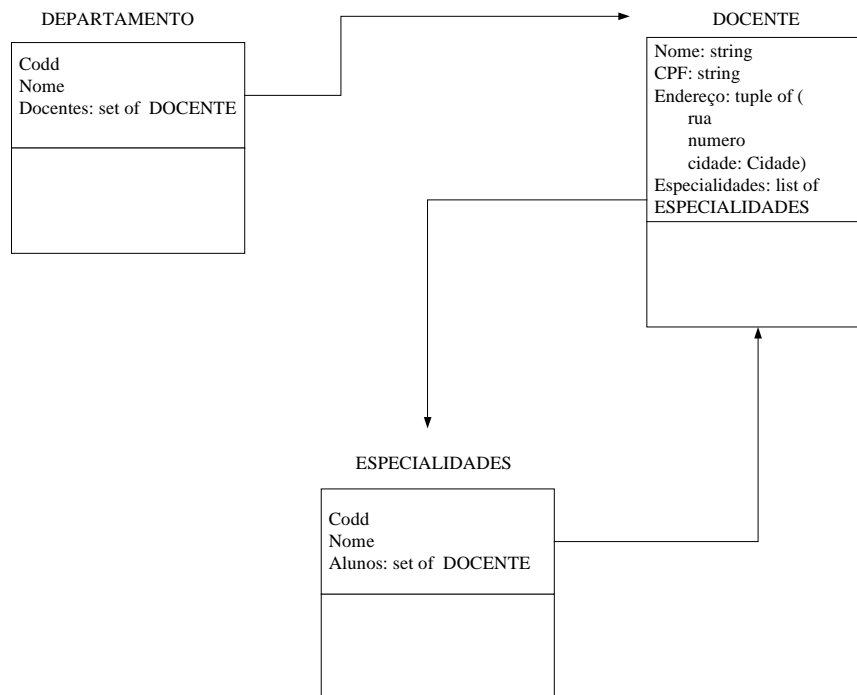


Figura 2-2. Exemplo de Objetos Complexos.

2.1.4. Herança

Herança é outro mecanismo de reusabilidade, onde uma classe é definida a partir da especialização de outras classes, herdando os atributos e métodos associados às mesmas. Estas classes definidas a partir de outras são chamadas de subclasses e as classes herdadas pela subclasse são conhecidas como superclasses. As superclasses são classes mais gerais que possuem propriedades comuns a diversas classes. Elas são uma generalização das subclasses. Portanto, a herança ocorre num relacionamento de generalização/especialização entre duas ou mais classes. Uma superclasse pode ser uma subclasse de outra superclasse e assim sucessivamente, formando uma hierarquia de classes.

Quando uma classe herda as propriedades de duas ou mais classes (superclasses) é dito que ela possui herança múltipla.

A Figura 2-3 apresenta um exemplo de relacionamento de herança entre as classes *Aluno*, *Pessoa* e *Funcionário*. As classes *Aluno* e *Funcionário* herdam os atributos e métodos da classe *Pessoa* (superclasse). A classe *Pessoa* é uma generalização das classes *Aluno* e *Funcionário* e estes últimos são especializações da classe *Pessoa*. O

relacionamento de herança é representado por uma seta espessa destacada partindo da subclasse para a superclasse. Forma-se assim uma hierarquia de classes.

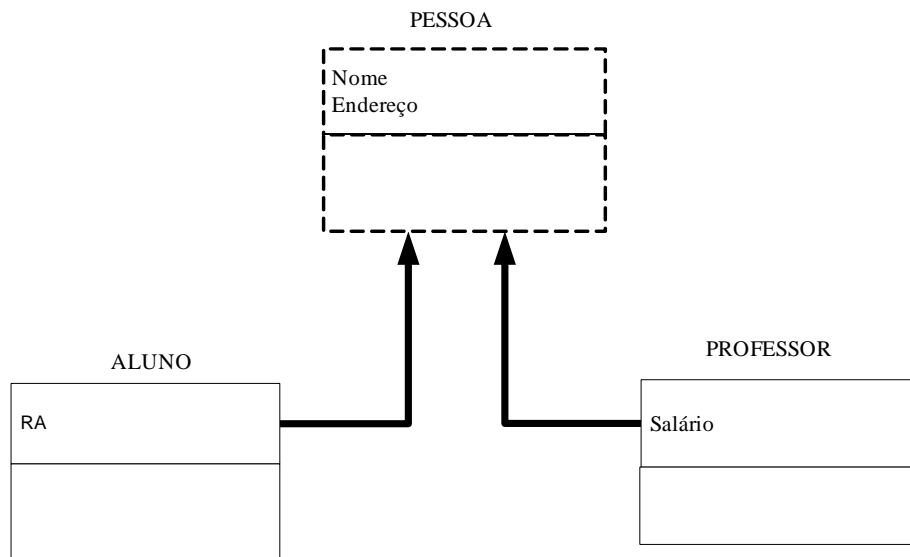


Figura 2-3. Exemplo de herança.

Em um BDOO, as consultas aplicadas a uma classe devem ser válidas na subclasse. Para tanto, a redefinição de propriedades ou deve ser proibida ou então permitida somente para restringir as propriedades das subclasses para domínios mais específicos.

3. XML e XML Schema

Este capítulo introduz as tecnologias XML e XML Schema e apresenta, de forma resumida, os recursos disponibilizados pela linguagem XML Schema, utilizada para descrever o esquema resultante do mapeamento proposto. No anexo 1 [MAU05], estes recursos da XML Schema são apresentados de forma detalhada e também é descrito o modelo de representação gráfica utilizada para apresentar os esquemas XML.

3.1.XML

A *XML* (eXtensible Markup Language) é uma linguagem de marcação, recomendada pela W3C (*World Wide Web Consortium*) [YER04] para descrever informações.

Esta linguagem está estruturada de forma a permitir flexibilidade na descrição de dados, já que não possui elementos pré-definidos, e é o usuário quem define o conjunto de elementos e suas características que melhor se aplicam no contexto para representação de seus dados. Por isso, a XML é considerada uma meta-linguagem.

A estrutura dos documentos XML é em forma de árvore hierárquica contendo marcações e dados. A Figura 3-1 mostra um exemplo de documento XML com dados a respeito de uma pessoa.

As marcações XML são conhecidas como elementos, como por exemplo o nome do exemplo da Figura 3-1. Cada elemento XML é delimitado por uma *tag inicial* (<nome>) e uma *tag final* (</nome>). O seu conteúdo pode ser outros elementos, um valor textual, ou ambos [MAR01].

Os elementos XML podem ainda conter atributos, que são pares nome-valor especificados na *tag* de início do elemento. O elemento *pessoa* do exemplo da Figura 3-1, possui um atributo *rg* (<pessoa rg="5.191.192-3">).

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
<peessoa rg="123.456.789-7">
  <nome>João</nome>
  <sobrenome>da Silva</sobrenome>
  <endereço>
    <rua>das Flores</rua>
    <numero>001</numero>
    <cep>85857-560</cep>
    <cidade>Florianópolis</cidade>
  </endereço>
</peessoa>
```

Figura 3-1. Documento XML.

A definição da XML foi motivada inicialmente pela necessidade de troca de informações na *Web*. Sua natureza auto-descritiva a torna uma opção eficaz para solucionar problemas de B2B (*Business to Business*) e extranet, viabilizando a comunicação entre aplicações diferentes.

Os documentos XML que possuem o objetivo de descrever informações de um mesmo tema, não necessariamente utilizam uma mesma estrutura para isto, já que as marcações de um documento não são pré-definidas pela linguagem XML. Quando é necessária a conformidade destes documentos com uma única estrutura, define-se, com o uso de uma linguagem de definição de esquemas XML, a estrutura dos documentos, e explicitamente define no documento XML que ele deve seguir esta estrutura definida. A linguagem mais utilizada para definição de esquemas de documentos XML é a *XML Schema (eXtensible Markup Language Schema)*.

3.2. XML Schema

A *XML Schema* consiste de uma linguagem de marcação no formato XML, recomendada pela W3C para definição de esquemas de documentos XML [W3C04].

Um esquema definido em XML Schema, esquema *XSD (XML Schema Definition)*, declara os elementos, sua ordem, restrições de conteúdo, atributos, grupos reusáveis, novos tipos de dados, ou seja, tudo que é necessário para estabelecer o que uma

instância de um documento XML, em conformidade com este esquema, pode ou deve ter.

Toda definição de um esquema está contida no elemento principal chamado *schema*.

As declarações existentes em um esquema podem ser divididas em dois grupos:

- grupo de gerenciamento;
- grupo de componentes de construção do modelo.

Este trabalho concentra-se no grupo de componentes de construção, visto que os dados do domínio de aplicação propriamente ditos são definidos através de seus elementos.

3.2.1. Recursos da linguagem XML Schema

A *XML Schema* é formada por um conjunto de elementos pré-definidos usados para definir esquemas de documentos XML. Estes elementos pré-definidos possuem atributos que permitem detalhar características como nome, uso (opcional ou obrigatório), valor *default*, entre outras, dos elementos e atributos definidos para o documento XML. A Figura 3-2 apresenta um exemplo de esquema XML que faz uso de alguns dos recursos disponibilizados pela linguagem.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
  <xs:element name="pessoa">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nome" type="xs:string"/>
        <xs:element name="sobrenome" type="xs:string"/>
        <xs:element name="endereço">
          <xs:complexType>
            <xs:sequence>
              <xs:element name= rua type= xs:string/
              <xs:element name= numero type= xs:int/>
              <xs:element name= cep type= string/>
              <element name= cidade type= xs:string/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name= rg type= string/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Figura 3-2. Um exemplo de esquema em XML Schema.

A seguir é apresentada uma tabela (Tabela 1) que descreve, de forma resumida, os principais recursos da linguagem XML Schema. Uma descrição mais detalhada pode ser encontrada no anexo 1 [MAU05].

Tabela 1. Resumo dos recursos da linguagem XML Schema

<pre> <schema> </schema> </pre>	<p>É o elemento raiz de qualquer documento XML Schema; Toda a definição de um esquema deve estar contida dentro deste elemento;</p>
<pre> <element > </element> </pre>	<p>Define os elementos de um documento XML; Possui um conjunto de atributos que permite definir as características de cada elemento como o nome, o tipo, seu número mínimo e máximo de ocorrências, se é abstrato ou não, entre outras características;</p>

<pre><complexType> </complexType></pre>	Define tipos de dados complexos que podem ser usados na criação de elementos ou de novos tipos; Assim como o elemento <i>element</i> , ele também possui vários atributos que fazem parte da sua definição e permitem detalhar as características deste tipo definido;
<pre><simpleType> </simpleType></pre>	Define tipos de dados simples que podem ser usados na definição de elementos, atributos ou de outros tipos;
<pre><union> </union></pre>	Define tipos de dados simples a partir da união de outros tipos simples cujos valores admitidos pelo tipo são qualquer um dos tipos incluídos na união; Os membros desta união é definido no atributo <i>memberType</i> ;
<pre><list/>:</pre>	Define tipos de dados simples que são listas de itens do tipo simples definido pelo atributo <i>itemType</i> ;
<pre><group> </group></pre>	Define um grupo de elementos; Quando se define um grupo de elementos é necessário especificar a ordem destes fazendo uso de indicadores de ordem (<i>sequence</i> , <i>all</i> , <i>choice</i>);
<pre><attributeGroup> </attributeGroup></pre>	Define um grupo de atributos;
<pre><attribute/></pre>	Define os atributos dos elementos; seu tipo deve ser simples;
<pre><sequence> </sequence></pre>	Indicador de ordem usado na definição de tipos complexos que estabelece que os elementos filhos deste tipo complexo devem aparecer na ordem em que foram declaradas;
<pre><choice> </choice></pre>	Outro indicador de ordem usado na definição de tipos complexos que estabelece que se deve escolher um dentre os elementos presentes em sua declaração;
<pre><all> </all></pre>	Outro indicador de ordem usado na definição de tipos complexos que estabelece que os todos os elementos presentes em sua declaração devem aparecer nas instâncias independente da ordem em que foram declarados;

<code><restriction></code> <code></restriction></code>	Define tipos derivados que restringe o conteúdo dos tipos que derivam, gerando, desta maneira, um novo tipo que é um subconjunto do tipo base;
<code><extension></code> <code></extension></code>	Define tipos derivados que estende o conteúdo do seu tipo base, gerando um novo tipo que herda as características do tipo base.

4. Trabalhos Relacionados

Algumas pesquisas têm sido realizadas para fornecer interoperabilidade de objetos entre sistemas heterogêneos através da XML. Contudo, os trabalhos relacionados enfatizam o mapeamento no sentido XML→OO e não se preocupam com o mapeamento no sentido inverso, que é o objetivo desta dissertação. Apesar disto, eles são relevantes para fins de estudo no que diz respeito à análise das abordagens de mapeamento XML-OO usadas.

Nas próximas seções são descritos quatro trabalhos relacionados que propõem formas de permitir a interoperabilidade entre XML e BDOO.

4.1.Proposta de Han et all

Este trabalho propõe um algoritmo para armazenar dados XML em BDOO/OR tomando como base a estrutura dos documentos XML [HAN03].

Através de regras de mapeamento adotadas, são inseridas no esquema XML informações de mapeamento e da estrutura do BD. Estas informações são utilizadas para gerar o esquema do BD para armazenamento dos dados dos documentos XML que possuem o esquema utilizado e também para catalogar as informações de mapeamento em classes geradas para este propósito no próprio BD. As informações presentes nestas classes catálogo são utilizadas pelo algoritmo que realiza o armazenamento dos dados no BD para identificar as características de cada componente mapeado.

As regras de mapeamento se baseiam numa análise realizada sobre a estrutura de documento XML e um esquema de BDOO/OR. Nesta análise é considerado que um documento XML é composto por elementos, atributos, relacionamentos entre um elemento e um atributo e relacionamentos entre elementos. Além disso, é considerado também que um esquema de BDOO/OR é composto por classes, atributos, relacionamentos entre uma classe e um atributo, e relacionamentos entre duas classes.

Com base nestas considerações, a Figura 4-1 a seguir, representa a correspondência existente entre os componentes de um documento XML e um esquema de BDOO/OR.

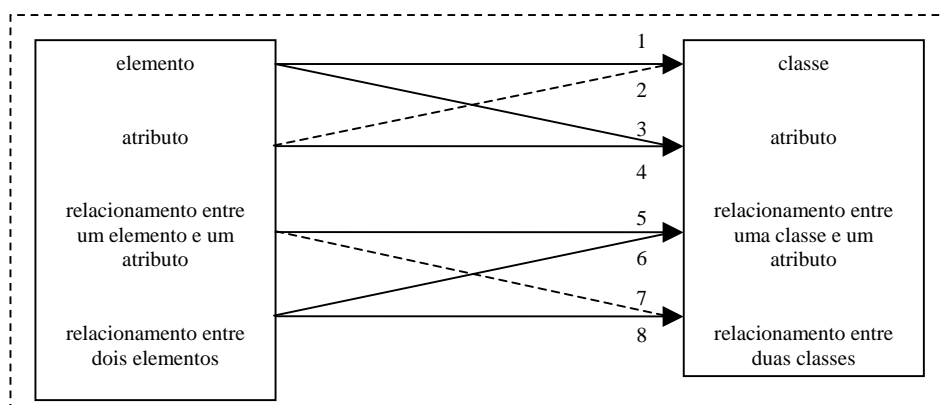


Figura 4-1. Possíveis mapeamentos de estruturas XML para um esquema de BDOO/OR [HAN03].

De acordo com as informações presentes na Figura 4-1 verifica-se que:

- os elementos de uma estrutura XML podem ser mapeados para uma classe ou um atributo;
- um atributo XML é mapeado para um atributo OO;
- os relacionamentos entre um elemento e um atributo são mapeados para um relacionamento entre uma classe e um atributo;
- os relacionamentos entre elementos podem ser mapeados para um relacionamento entre uma classe e um atributo ou então para um relacionamento entre duas classes.

Estas regras de mapeamento são utilizadas para gerar um XML Schema contendo, além da estrutura do documento XML, as informações de mapeamento entre os componentes do documento XML e o BDOO/OR e o esquema do BD. Para isso, foram adicionados três novos sub-elementos ao elemento *appinfo* pertencente à linguagem XML Schema. Dois dos sub-elementos são *class* e *column*, que possibilitam que as classes e os atributos do esquema do BD sejam especificados e também possibilitam o mapeamento dos elementos e/ou atributos da XML para classes ou atributos OO. O sub-elemento *class* possui o atributo *name* para especificar o nome da classe e o sub-elemento *column* possui os atributos *name* e *type* para especificar, respectivamente, o nome e o tipo do atributo.

O terceiro sub-elemento é o *Relationship*, que especifica o mapeamento dos relacionamentos entre dois elementos para um relacionamento entre duas classes. Os

relacionamentos são classificados em um-para-um ou um-para-muitos. Com base nas restrições de cardinalidade, eles também são classificados em uni-direcional ou bi-direcional.

Para especificar o mapeamento de relacionamentos, o elemento *Relationship* possui quatro atributos: *parent*, *child*, *cardinality* e *isOrdered*. O atributo *parent* é usado para indicar o elemento pai do relacionamento entre dois elementos e o atributo *child* para especificar o elemento filho do mesmo relacionamento. O atributo *cardinality* especifica as restrições de cardinalidade do relacionamento e o atributo *isOrdered* especifica as restrições para preservação de ordem. A direção do relacionamento é uni-direcional se o atributo *child* é omitido. Caso contrário é bi-direcional. O exemplo a seguir, apresenta um trecho de um do XML Schema contendo as informações de mapeamento. O esquema apresentado na Figura 4-2(a) possui um elemento global (*book*) com dois elementos filhos: um simples (*title*) e um complexo (*author*), e um atributo (*id*). A Figura 4-2(b) mostra um trecho do mesmo esquema com as informações sobre o mapeamento inseridas. No exemplo, o elemento *book* será mapeado para uma classe. O sub-elemento *title* e o atributo *id* para atributos desta classe. O elemento *author* é mapeado para uma classe, visto que é complexo. Nas informações sobre o mapeamento são especificados os atributos de referências necessários para estabelecer o relacionamento entre as classes *book* e *autor*. Na classe *book* a referência especifica que será uma lista de *author*, devido ao indicador de ocorrência do mesmo ser *0..n*.

Uma vez descrito o mapeamento, essas informações são armazenadas no BDOO/OR em três classes catálogos específicos para esse fim: *xmlSysElementst*, *xmlSysAttributes* e *xmlSysRelationships*. A primeira tem por objetivo armazenar informações dos elementos mapeados. A segunda, informações sobre os atributos mapeados e a última informações sobre os relacionamentos entre os elementos mapeados. A Figura 4-3 apresenta a estrutura destas três classes.

Por fim, este trabalho apresenta um algoritmo para armazenamento dos dados de documentos XML em BDOO/OR. Este algoritmo recebe como entrada um documento XML onde todos os elementos são verificados um a um. Se o elemento é mapeado para uma classe então cria uma instância da classe e estabelece o relacionamento deste com algum outro elemento que seja seu pai. Caso o elemento seja mapeado para um atributo, seu valor é armazenado no atributo do objeto. E por último, todos os atributos do

elemento são armazenados nos atributos da classe. A Figura 4-4 a seguir apresenta o algoritmo proposto pelo trabalho.

```

<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
  <xsd:annotation>
    <xsd:documentation>
      Book schema for www.book.com.
      Copyright 2001 www.book.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>

```

(a)

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:appinfo>
      <Class name="book">
        <Column name="book.authors" type="list(ref(author))"/>
        <Column name="book.id" type="integer"/>
        <Column name="book.title" type="varchar(100)"/>
      </Class>
      <Class name="author">
        <Column name="author.book" type="ref(book)"/>
        <Column name="author.name" type="varchar(100)"/>
        <Column name="author.email" type="varchar(100)"/>
      </Class>
      <Relationship parent="book.authors" child="author.book"
        cardinality="oneToMany" isOrdered="yes"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:element name="book">
    <xsd:annotation>
      <xsd:appinfo>
        <Class name="book"/>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>
              <Column name="book.title"/>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

```

informações do esquema do banco de dados

mapeamento do relacionamento entre os elementos *books* e *author*

mapeamento do elementos *book*

mapeamento do elemento *author*

(b)

Figura 4-2. XML Schema (a), trecho XML Schema contendo informações sobre o mapeamento (b) [HAN03].

xmlSysElements

name	elementId	elementName	flag	classId	columnNo
type	integer	varchar	char	integer	integer

xmlSysAttributes

name	elementId	attributeNo	attributeName	classId	columnNo
type	integer	integer	varchar	integer	integer

xmlSysRelationships

name	parentId	childId	cardinality	flag	isOrdered	parentClassId	parentColumnNo	childClassId	childColumnNo
type	integer	integer	char	char	char	integer	integer	integer	Integer

Figura 4-3. Classes catalogo para armazenar informações sobre o mapeamento[HAN03].**Algoritmo StoreXML_ORDB****Entrada:**

D: documento XML

```

1:  for (elemento E no documento XML D)
2:  {
3:    if (E é mapeado para uma classe)
4:    {
5:      Cria um objeto O na classe para o qual E é mapeado;
6:      if (existe um objeto Op que armazena o elemento pai e o relacionamento R)
7:      {
8:        if (R é um relacionamento um-para-um)
9:          Armazena o OID de O no atributo de Op que é uma referencia a O;
10:       else if (R é um relacionamento um-para-muitos)
11:          Armazena o OID de O no atributo de Op que é uma coleção de referencias para O;
12:       if (R é um relacionamento bi-direcional)
13:          Armazena o OID de O no atributo de Op que é uma referencia para Op;
14:      }
15:    }
16:    else if (E é mapeado para uma coluna C)
17:    {
18:      if (o tipo de C é simples)
19:        Armazena o valor de E num atributo do objeto O para o qual E é mapeado
20:      else if (o tipo de C é uma coleção de simples)
21:        Armazena o valor de A num atributo do objeto O para o qual E é mapeado como uma coleção de elementos
22:    }
23:    for (cada atributo A pertencente a E)
24:      Armazene o valor de A num atributo do objeto O para o qual E é mapeado
25:  }

```

Figura 4-4. Algoritmo para armazenar documentos XML em um BDOO/OR [HAN03].

4.2. Proposta de Renner

Este trabalho apresenta uma abordagem para armazenar dados XML em SGBDOO fazendo uso de uma especificação Java – JAXB (*Java Architecture for XML Binding*) que define uma arquitetura para comunicação entre tecnologias Java e documentos XML [REN01]. A especificação desta arquitetura está disponível em <http://java.sun.com/xml/jaxb/index.jsp>. Um dos objetivos da JAXB é mapear dados XML a partir de XML Schema para objetos Java e vice-versa. A Figura 4-5 apresenta os elementos envolvidos no mapeamento de documentos XML para objetos Java.

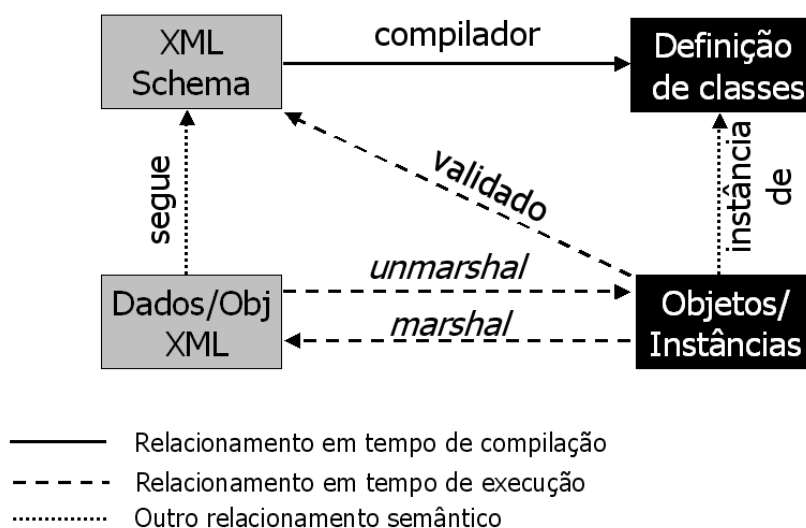


Figura 4-5. Mapeamento de XML para objetos Java [REN01].

Esta arquitetura possui um compilador para gerar classes em Java a partir de XML Schemas. Os dados das instâncias XML são instanciados para objetos Java a partir do método *unmarshal* e então pode-se fazer uso das tecnologias desenvolvidas para Java para operar com um BDOO, como armazenar dados e realizar consultas. O processo inverso, gerar instâncias XML a partir de objetos Java é feito utilizando o método *marshal*.

A arquitetura JAXB tira proveito dos conceitos implícitos na linguagem XML Schema para definição de dados que possuem uma correspondência direta com os conceitos da Orientação a Objetos. O compilador recebe como entrada um XML

Schema contendo as definições de tipos de dados e gera classes em Java para cada tipo definido como apresentado na Figura 4-6.

```
<xsd:complexType name= "Address">
  <xsd:sequence>
    <xsd:element name= "street" type= "xsd:string"/>
    <xsd:element name= "street" type= "xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

(a)

```
<complexType name= "USAddress">
  <complexContent>
    <extension base= "Address">
      <sequence>
        <element name= "state" type= "UState"/>
        <element name= "zip" type= "positiveInteger"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

(b)

```
Class Address
{
private String street;
private String city;

public String getStreet();
public void setStreet(String);
public String getCity();
public String setCity(String);
}

Class USAddress
{
private String state;
private String zip;

public String getState();
public void setState(String);
public String getZip();
public String setZip(int);
}
```

(c)

Figura 4-6. Mapeamento de tipos de dados XML para objetos Java: definição de um tipo complexo(a), definição de tipo derivado(b), classe Java gerada(c) [REN01].

Para cada tipo de dado do XML Schema, o compilador gera um arquivo fonte contendo uma classe Java. Cada classe Java contém campos para cada atributo ou elemento do tipo de dado do XML.

Neste trabalho, documentos XML são classificados em dados XML e documentos XML conforme as características presentes na sua estrutura. Dados XML são documentos definidos para processamento eletrônico, enquanto documentos XML são

definidos para pessoas escreverem ou lerem algo, não havendo muito interesse em manipular computacionalmente as informações presentes no mesmo. Desta forma, documentos de dados XML são formados por um conjunto de dados bem estruturados e bem definidos que podem ser mais facilmente manipulados para extração de informações úteis a aplicações, justificando seu armazenamento em um BD. O JAXB lida com este tipo de documento.

Existem BDOOs que geram o esquema do BD analisando os arquivos .class gerados pela linguagem Java. Desta forma, a integração de JAXB com estes BDs permite a geração de esquemas OO a partir de esquemas XML.

4.3. Proposta de Chung et al

Esta abordagem se preocupa com o processo de armazenamento dos dados de documentos XML em um BDOO tomando como base informações da estrutura do XML contidas em uma DTD (*Document Type Definition*), linguagem com menos robustez, que permite definir esquemas de documentos XML.[CHU01].

O processo de geração do esquema do BDOO para armazenar os dados de instâncias XML é realizada em duas etapas:

- aplica-se a técnica *inlining* desenvolvida por [SHA99] para BDs Relacionais, contendo algumas adaptações a fim de obter classes que englobam o maior número de elementos possíveis;
- classifica-se os elementos da DTD e reconstroem as classes usando herança.

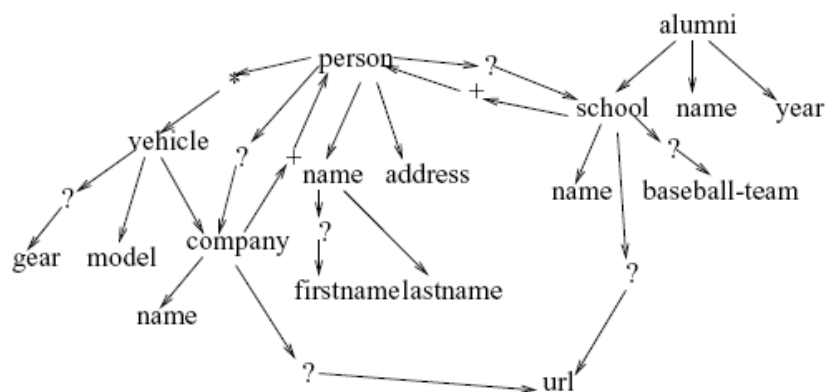
A técnica *inlining* consiste em representar a DTD num grafo como mostra a Figura 4-7 e analisar cinco considerações sobre ela para definir a estrutura do esquema do BDOO.

```

<!ELEMENT person (name, address, vehicle*,(school|company))>
<!ELEMENT name (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT vehicle (model, company, gear?)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT gear (#PCDATA)>
<!ELEMENT school (name, baseball-team?, person+,url?)>
<!ATTLIST school name CDATA #CDATA REQUIRED>
<!ELEMENT baseball-team (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT company (name, person+, url?)>
<!ATTLIST company name CDATA #CDATA REQUIRED>
<!ELEMENT alumni (name, year, school)>
<!ATTLIST alumni name CDATA #CDATA REQUIRED>
<!ELEMENT year (#PCDATA)>

```

(a)



(b)

Figura 4-7. Exemplo de DTD (a), representação em grafo da DTD(b) [CHU01].

A seguir são apresentadas as considerações estabelecidas [CHU01]:

1. Cria-se classes para elementos que são de grau zero;
2. Elementos que possuem uma cardinalidade de * ou um + também originam classes;
3. Nodos com grau um são englobados pelo nodo pai;
4. Entre elementos mutuamente recursivos, onde todos possuem grau um, um deles vira uma relação separada;
5. Elementos tendo um grau maior do que um, viram relações separadas.

O esquema do BDOO gerado após a aplicação destas regras para o exemplo apresentado Figura 4-7 é apresentado na Figura 4-8.

```

class Person public type tuple(name.firstname:string,name.lastname:string,
    address:string,vehicle:list(Vehicle),school:School,company:Company)
class School public type tuple(name:string,baseball-team:string,
    person:list(Person),url:Url)
class Alumni public type tuple(name:string, year:String,school:School)
class Company public type tuple(name:string,person:list(Person),url:Url)
class Url inherit Text
class Vehicle public type tuple(model:string,company:Company,gear:string)

```

Figura 4-8. Um esquema para o BDOO [CHU01].

A segunda etapa consiste em tentar tirar proveito do conceito de herança existente em OO para gerar o esquema do BDOO, para isso cada elemento que foi mapeado para uma classe é abstraído para uma expressão regular ou para um elemento tipo PCDATA, como mostra a Figura 4-9(a). Sobre estas expressões regulares são construídas expressões regulares relaxadas para extrair somente as informações necessárias na classificação dos elementos (Figura 4-9(b)).

person: (name, address, vehicle, (school|company))*

(a)

person : (name, address, (vehicle|⊥), (school|company))

(b)

Figura 4-9. Uma expressão regular(a), expressão regular relaxada da expressão regular(b) [CHU01.]

Estas expressões regulares relaxadas, são então, as entradas de um algoritmo (Figura 4-10) que gera um autômato para representar os possíveis conjuntos de dados que podem surgir a partir da declaração da DTD. A Figura 4-11 apresenta o autômato gerado para o exemplo da Figura 4-9. Para cada possibilidade é gerada uma classe distinta fazendo uso da herança, como mostra a Figura 4-10.

- 1: **Input:** A relaxed regular expression r
- 2: **Output:** An automaton M
- 3: **procedure** Make_DTD_Automata(regular expression r)
- 4: **if** $r = a$ ($a \in \Sigma$) **then**
- 5: Construct an automaton M as shown in Figure 5;
- 6: return M ;
- 7: **else if** $r = r_1|r_2$ **then**
- 8: $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1) \leftarrow \text{Make_DTD_Automata}(r_1)$;
- 9: $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2) \leftarrow \text{Make_DTD_Automata}(r_2)$;
- 10: Construct the new automaton $M = (Q_1 - \{q_1\} \cup Q_2 - \{q_2\}, \Sigma_1 \cup \Sigma_2, \delta, [q_1, q_2], F_1 \cup F_2)$ from the automata M_1 and M_2 , where δ is defined by
 1. $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - \{q_1\}$ and $a \in \Sigma_1$,
 2. $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - \{q_2\}$ and $a \in \Sigma_2$,
 3. $\delta([q_1, q_2], a) = \delta_1(q_1, a)$ where $a \in \Sigma_1$,
 4. $\delta([q_1, q_2], a) = \delta_2(q_2, a)$ where $a \in \Sigma_2$;
- 11: **else** $\{ r = r_1, r_2 \}$
- 12: $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1) \leftarrow \text{Make_DTD_Automata}(r_1)$;
- 13: $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2) \leftarrow \text{Make_DTD_Automata}(r_2)$;
- 14: Let the final states F_1 of M_1 be states f_1, f_2, \dots, f_m ($m \geq 1$).
Construct the new automaton $M = (Q_1 - F_1 \cup Q_2 - \{q_2\} \cup \{[f_1, q_2], [f_2, q_2], \dots, [f_m, q_2]\}, \Sigma_1 \cup \Sigma_2, \delta, q_1, F_2)$ from the automata M_1 and M_2 , where δ is defined by
 1. $\delta(q, a) = \delta_1(q, a)$ for $q \in Q_1 - F_1$, $\delta_1(q, a) \neq f_k$ (where $1 \leq k \leq m$), and $a \in \Sigma_1$,
 2. $\delta(q, a) = \delta_2(q, a)$ for $q \in Q_2 - q_2$ and $a \in \Sigma_2$,
 3. $\delta([f_k, q_2], a) = \delta_2(q_2, a)$ for all k (where $k = 1, 2, \dots, m$) and $a \in \Sigma_2$,
 4. $\delta(q_f, a) = [f_k, q_2]$ for all q_f which satisfies $\delta_1(q_f, a) = f_k$ (where $1 \leq k \leq m$) and $a \in \Sigma_1$;
- 15: **end if**
- 16: return M

Figura 4-10. Algoritmo de construção de um autômato [CHU01].

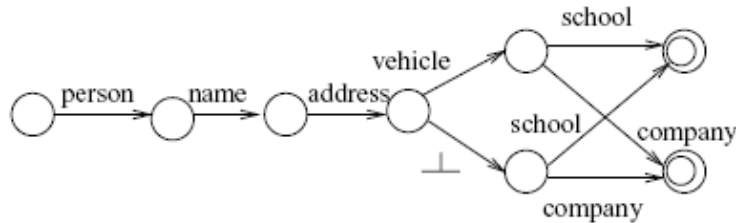


Figura 4-11. Um autômato para um trecho da DTD [CHU01].

```

class Person public type tuple(name.firstname:string,name.lastname:string,
    address:string)
class Person1 inherit Person type tuple(vehicle:list(Vehicle),
    school:School)
class Person2 inherit Person type tuple(vehicle:list(Vehicle),
    company:Company)
class Person3 inherit Person type tuple(school:School)
class Person4 inherit Person type tuple(company:Company)
...

```

Figura 4-12. Esquema para o BDOO gerado após aplicação da segunda etapa da proposta [CHU01].

4.4. Proposta de Salim et al

Este trabalho apresenta um conjunto de regras de mapeamento para transformar esquemas XML para um modelo OO com o objetivo de gerar uma documentação do mesmo, facilitando então sua manutenção [SAL04]. Para isso, faz-se uso dos recursos disponibilizados pela linguagem de modelagem UML buscando como resultado diagramas UML para representá-lo. A Tabela 2 apresenta o conjunto resumido das regras definidas.

De acordo com a Tabela 2, os elementos declarados num esquema podem ser mapeados para classes ou atributos de classes dependendo da sua localização. Por exemplo, um elemento global é mapeado para uma classe, no entanto se ele é local é mapeado para um atributo. Os atributos do esquema também são analisados quanto a sua localização para definir o mapeamento, sendo um atributo local mapeado para um atributo da classe e um global para outra classe. Os tipos de dados simples definidos no esquema, assim como os elementos e atributos, têm um tratamento distinto dependendo de onde estão declarados. Se for global, é mapeado para classe; caso contrário, é mapeado para atributo. Já o tratamento para novas declarações de tipos de dados complexos não é apresentado uma regra de mapeamento caso ele seja global, mas é dada ênfase no construtor utilizado para defini-los (*sequence*, *all* ou *choice*). Se for utilizado o construtor *sequence* ou *choice*, são mapeados para atributos e se for utilizado o construtor *all*, os elementos filhos obedecem as regras de mapeamento definidas para elementos.

Tabela 2. Resumo das regras de mapeamento dos construtores de XML Schema [SAL04].

Construtores XML Schema		Construtores UML	
Espaço de nomes		Pacote	
Declaração de elementos	Elemento global	Classe e propriedades {root} (se necessário)	
	Elemento local	Atributo	
	Restrições de ocorrência (minOccurs, maxOccurs)	Multiplicidade de atributo	
	Restrição fixed	Restrição {frozen}	
	Restrição default	Restrição {changeable}	
	Nullable	Restrição individual {pode ser NULL}	
Declaração de atributo	Atributo global	Classe	
	Atributo local	Atributo	
	Restrição Fixed	Restrição {frozen}	
	Restrição Default	Restrição {changeable}	
	Restrição Optional	Multiplicidade [0..1]	
	Restrição Required	Multiplicidade [1..1]	
Definição de Tipo Simples	Tipo Simples global	Classe e esteriótipo <<type>> (se necessário)	
	Tipo Simples local	Atributo	
	Restrição		Atributo
		Construtor filho	Restrições individuais do atributo
		Tipo base	Tipo do atributo
		enumeration	Tipo de dado enum do C++
	Tipo de dado List	Tipo de dado list do C++	
	Tipo de dado Union	Tipo de dado union do C++	
Definição de Tipo Complexo	Compositor sequence	Tipo de dado struct do C++ ou executar {ordered} em compartimentos de atributos	
	Compositor choice	Tipo de dado union do C++	
	Compositor all	Transformar cada elemento filho os papéis para transformar elementos	
	Conteúdo simples	Seguir a transformação de papéis para tipos simples. Atributos XML Schema adicionais serão transformados para atributos UML	
	Conteúdo vazio	Seguir os papéis para transformação de elementos de tipos complexos	
	Conteúdo misto	Uma classe com atributos string artificiais e ou uma execução de restrição ou uma nota UML especificando a que classe UML deve conter texto ou string	
	Derivado por extensão	Subclasses com relacionamento de generalização para a classe que representa o tipo base do tipo derivado do XML Schema	
	Derivado por restrição	Uma nova classe	
Grupo de Substituição		Union do C++ ou classes associadas com a restrição {xor}	
Modelo de grupo		Não é transformado. Papéis de substituição de textos é usado se necessário	
Grupo de atributo		Não é transformado. Papéis de substituição de textos é usado se necessário	
Definição de restrição de identidade		Qualificados da associação entre a classe qualificadora e a classe alvo	
	Elemento Key	Classe qualificadora	
	Elemento Keyref	Classe alvo	
Declaração de Notation		Nota	
Construtores XML Schema auxiliares		Nota	

5. Abordagem de Mapeamento BDOO-XML Schema

Este capítulo apresenta a abordagem proposta para o mapeamento BDOO→XML Schema. Esta abordagem descreve um conjunto de regras de mapeamento e um algoritmo para aplicação destas regras. O conjunto de regras de mapeamento define correspondências entre conceitos do modelo lógico de BDOO e estruturas da XML Schema. Já o algoritmo estabelece uma abordagem de aplicação das regras definidas para obter um esquema em XML Schema que represente o esquema de um BDOO qualquer.

5.1.Regras de Mapeamento

Esta seção contém a definição de um conjunto de regras de mapeamento dos conceitos existentes em BDOO para XML Schema. Para isso, foram estudados os conceitos do modelo de BDOO que definem a estrutura de classes de objetos e definida uma representação destes conceitos no modelo da XML Schema.

A partir de possíveis estruturas encontradas em esquemas de BDOO, são definidas regras de mapeamento para estruturas correspondentes em esquemas XML Schema.

De forma geral, um esquema de BDOO é composto por definições de classes concretas e abstratas com estruturas compostas por atributos e métodos, e relacionamentos de herança entre elas. Este trabalho busca estruturas equivalentes em XML para mapeamento dos dados presentes numa classe e não se preocupa com os métodos¹.

A Tabela 3 apresenta um resumo do conjunto de regras definidas.

¹ Caso for desejado armazenar métodos em documentos XML, eles poderiam ser especificados dentro de um elemento *annotation* que tem o propósito de armazenar informações adicionais ou instruções de processamento para serem utilizadas por um software qualquer.

Tabela 3. Regras de mapeamento de esquemas BDOO para esquemas XML.

BDOO		XML Schema	
Classe	Concreta	Tipo Complexo com indicador de ordem <i>sequence</i> ou <i>all</i>	
	Abstrata	Tipo Complexo abstrato com indicador de ordem <i>sequence</i> ou <i>all</i>	
Atributo	Primitivo	Atributo ou Elemento Simples	
	Complexo ²	Tupla	Elemento Complexo com indicador de ordem <i>sequence</i> ou <i>all</i>
		Lista	Elemento Simples ou Complexo com a cardinalidade $0..n$
		Conjunto	Elemento Simples ou Complexo com cardinalidade $0..n$
Referência ³	Atributo tipo ID no elemento que está sendo referenciado e atributo tipo IDREF para o atributo que estabelece a referência		
Herança	Inserção de uma restrição na classe especializada, se na herança houve apenas redefinição de atributos; ou inserção de uma extensão na classe especializada, se na herança houve apenas adição de novos atributos; ou ainda, definição de um Elemento Complexo auxiliar que faz uma restrição sobre a classe generalizada e inserção de uma extensão da classe auxiliar na classe especializada, se na herança houve adição de novos atributos e redefinição de outros;		
Atributos Redefinidos	Subgrupo do atributo derivado		
Herança Múltipla	Inclusão, na classe especializada, dos componentes gerados pelas regras aplicadas aos atributos de cada classe herdada, se os atributos não foram redefinidos na classe especializada.		

² São considerados atributos complexos aqueles com domínio estruturado, definido através de construtores de tipos.

³ São considerados atributos de referência aqueles cujo domínio é outro objeto.

A seguir, são apresentadas, de forma detalhada, as regras de mapeamento apresentadas na Tabela 3 e exemplos das mesmas. Os exemplos são apresentados sobre a representação gráfica adotada para os esquemas.

A descrição das regras de mapeamento é feita utilizando a seguinte notação:

- Definição descritiva do conceito do modelo OO;
- Conceitos do modelo XML Schema correspondente ao conceito OO descrito anteriormente.

Os conceitos do modelo XML Schema que devem ser gerados para o mapeamento são apresentados em itens. Quando existe mais de uma opção de mapeamento para um mesmo conceito de OO em XML Schema, estas opções são apresentadas dentro de chaves e é feito o uso da palavra *ou* para deixar claro que são itens exclusivos. Caso haja necessidade de mais de um item para descrever o que deve ser definido em XML Schema para o conceito OO ser mapeado, estes são apresentados em itens separados e é feito o uso da palavra *e* para deixar explícito que as duas ações devem ser tomadas.

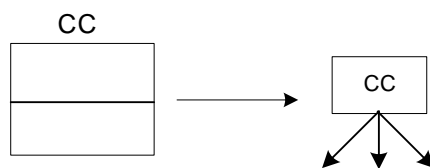
5.1.1. Regra CC – Mapeamento de Classe Concreta

Em XML, uma estrutura equivalente à estabelecida por uma classe através de seu conjunto de atributos é definida através de tipos complexos contendo sub-elementos ou atributos correspondentes aos atributos da classe. Portanto, no mapeamento proposto, toda classe concreta definida num esquema do BDOO tem um tipo complexo correspondente declarado no XML Schema usando ou o indicador de ordem *sequence* ou *all* de acordo com a preferência.

Definição: Uma *classe instanciável* de nome *CC* definida no esquema de um BDOO gera:

- {
 - Um *tipo complexo* de nome *CC* com indicador de ordem *sequence* no esquema XML; ou
 - Um *tipo complexo* de nome *CC* com indicador de ordem *all* no esquema XML.

Exemplo:



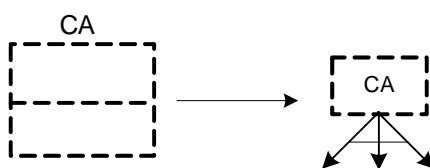
5.1.2. Regra CA – Mapeamento de Classe Abstrata

As classes abstratas do esquema são estruturas idênticas às classes concretas. Portanto, para o mapeamento, toda classe abstrata definida num esquema do BDOO, também tem um tipo complexo correspondente declarado no XML Schema. Entretanto, o atributo *abstract* do tipo complexo correspondente é definido como *true*.

Definição: Uma *classe abstrata* de nome *CA* definida no esquema de um BDOO gera:

- Um *tipo complexo abstrato* de nome *CA* com indicador de ordem *sequence* no esquema XML; ou
- Um *tipo complexo abstrato* de nome *CA* com indicador de ordem *all* no esquema XML.

Exemplo:



5.1.3. Regra TP – Mapeamento de Tipo Primitivo

Os atributos declarados numa classe que correspondem a tipos primitivos e as estruturas complexas, como conjunto, lista ou tupla que possuem tipos primitivos, são mapeados para sub-elementos simples de tipos nativos equivalentes aos definidos para o atributo na classe ou então para atributos em XML no tipo complexo correspondente a

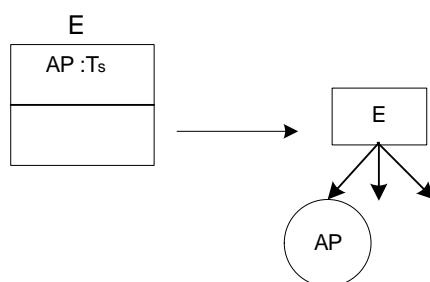
classe ou estrutura complexa do atributo. Estas alternativas podem ser escolhidas de acordo com a preferência, não havendo condições ligadas a elas.

Definição: Um *atributo primitivo* AP de tipo T_s pertencente à *classe* ou *estrutura complexa* E , definida no esquema de um BDOO gera:

- Um elemento de nome AP e tipo T_s , filho do elemento complexo ou tipo complexo correspondente a E no esquema XML; ou
- Um atributo de tipo T_s e nome AP pertencente ao elemento complexo ou tipo complexo correspondente a E no esquema XML.

Observação: O mapeamento dos tipos de dados primitivos de um BDOO para os tipos primitivos da XML Schema deve ser validado posteriormente pelo usuário.

Exemplo:



O exemplo mostrado acima ilustra o mapeamento de um atributo primitivo de uma classe E para um elemento filho do tipo complexo correspondente a E .

5.1.4. Regra ETT – Mapeamento de Estrutura Tipo Tupla

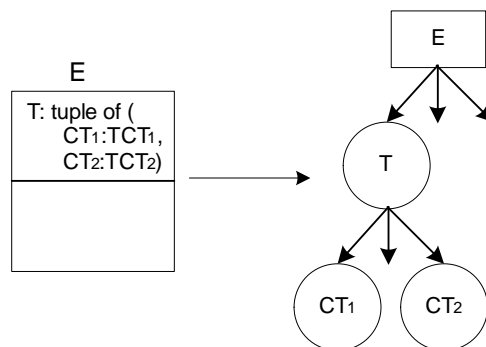
Uma estrutura tipo tupla define seu conteúdo como um conjunto de outros atributos de tipos distintos, ou seja, é uma estrutura complexa formada por um conjunto de outras estruturas (que pode ser complexa ou não). Em esquemas XML, a estrutura equivalente consiste em um subelemento complexo contendo outros sub-elementos que correspondam aos atributos pertencentes à tupla. Os sub-elementos correspondentes aos atributos da tupla são gerados a partir das regras de mapeamento definidas para seu tipo, que podem ser primitivos, tuplas, listas, conjuntos ou então de referência. Por exemplo,

se um atributo da tupla for de tipo primitivo, este deve ser gerado pela regra de mapeamento de tipos primitivos.

Definição: Uma estrutura do tipo tupla ET com um conjunto de atributos componentes de nomes CT_1, CT_2, \dots, CT_n e domínios $TCT_1, TCT_2, \dots, TCT_n$, respectivamente, pertencente à classe ou estrutura complexa E gera:

- Um elemento complexo de nome T com indicador de ordem *sequence*, filho do elemento complexo ou tipo complexo correspondente a E ; ou
 - Um elemento complexo de nome T com indicador de ordem *all* filho do elemento complexo ou tipo complexo correspondente a E ,
- , se T é o domínio de um atributo de E ;
- e
- Estruturas filhas de T , geradas pelas regras de mapeamento para os domínios $TCT_1, TCT_2, \dots, TCT_n$ aplicadas nos componentes de nomes CT_1, CT_2, \dots, CT_n ,
- , se T é o domínio de um atributo de E ;
- Estruturas filhas de E , geradas pelas regras de mapeamento para os domínios $TCT_1, TCT_2, \dots, TCT_n$ aplicadas nos componentes de nomes CT_1, CT_2, \dots, CT_n ;

Exemplo:



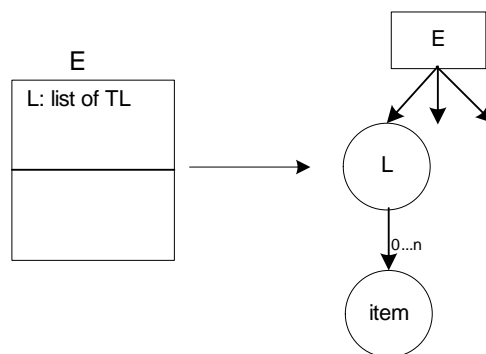
Este exemplo possui uma classe chamada de E com um atributo tipo tupla de nome T . A tupla T possui dois atributos de tipos primitivos.

5.1.5. Regra ETL – Mapeamento de Estrutura Tipo Lista

As estruturas do tipo lista correspondem a uma coleção de dados primitivos ou não, ordenados. A linguagem XML fornece suporte para especificação de listas somente para tipos simples que sejam diferentes de cadeia de caracteres. Para listas de estruturas complexas, a estrutura em XML é definida em dois passos: (i) define-se um elemento complexo filho da estrutura que contém a lista (levando em consideração que podem existir estruturas complexas formadas por outras estruturas complexas) ou do tipo complexo correspondente a classe. Este passo só ocorre se a estrutura lista é o domínio de um atributo da estrutura ou tipo complexo; (ii) define-se um subelemento nomeado de *item* contendo uma estrutura equivalente definida pela regra de mapeamento para o tipo desta lista com os indicadores de ocorrência *minOccurs* e *maxOccurs* definidos para 0 e *unbounded*, respectivamente. O conceito de ordem estabelecido por uma lista é implícito a organização de seus itens em XML.

Definição: Uma estrutura tipo lista *EL* de um tipo *TL* pertencente à classe ou estrutura complexa *E* gera:

- Um elemento simples de nome L que contém uma lista do tipo TL filho do elemento complexo ou tipo complexo correspondente a E , se TL corresponde a um tipo de dado atômico diferente de cadeia de caracteres; ou
 - gera-se:
 - Um elemento complexo de nome L com indicador de ordem *sequence* filho do elemento complexo ou tipo complexo correspondente a E ; ou
 - Um elemento complexo de nome L com indicador de ordem *all* filho do elemento complexo ou tipo complexo correspondente a E , se L é o domínio de um atributo de E ;
- e
 - Um elemento de nome *item*, filho do elemento L , com uma estrutura gerada pela regra de mapeamento para o tipo TL , com indicadores de ocorrência *minOccurs* e *maxOccurs* definidos respectivamente para “0” e “unbounded”
- , se L é o domínio de um atributo de E ; ou
 - Um elemento de nome *item*, filho de E , com uma estrutura gerada pela regra de mapeamento para o tipo TL , com indicadores de ocorrência *minOccurs* e *maxOccurs* definidos

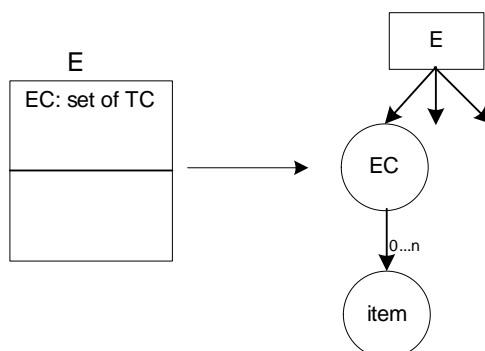
Exemplo:

5.1.6. Regra ETC – Mapeamento de Estrutura Tipo Conjunto

Estas estruturas, assim como as estruturas de lista, correspondem a uma coleção de dados primitivos ou não, porém, não são ordenados. Para mapear estas estruturas para XML também são utilizados dois passos: (i) define-se um elemento complexo filho da estrutura que contém o conjunto (levando em consideração que pode existir estruturas complexas formadas por outras estruturas complexas) ou do tipo complexo correspondente a classe. Este passo só ocorre se a estrutura conjunto é o domínio de um atributo da estrutura ou tipo complexo. (ii) define-se um subelemento nomeado de *item* contendo uma estrutura equivalente definida pela regra de mapeamento para o tipo deste conjunto com os indicadores de ocorrência *minOccurs* e *maxOccurs* definidos para 0 e *unbounded*, respectivamente.

Definição: Uma estrutura tipo conjunto *EC* de um tipo *T* pertencente à classe ou estrutura complexa *E* gera:

- {
 - Um elemento complexo de nome *C* com indicador de ordem *sequence* filho do elemento complexo ou tipo complexo correspondente a *E*; ou
 - Um elemento complexo de nome *C* com indicador de ordem *all* filho do elemento complexo ou tipo complexo correspondente a *E*;
 , se *C* é o domínio de um atributo de *E*;
- e
- {
 - Um elemento de nome *item*, filho do elemento *C*, com uma estrutura gerada pela regra de mapeamento para o tipo *TC*, com indicadores de ocorrência *minOccurs* e *maxOccurs* definidos respectivamente para “0” e “*unbounded*”, se *C* é o domínio de um atributo de *E*; ou
 - Um elemento de nome *item*, filho de *E*, com uma estrutura gerada pela regra de mapeamento para o tipo *TC*, com indicadores de ocorrência *minOccurs* e *maxOccurs* definidos respectivamente para “0” e “*unbounded*”;

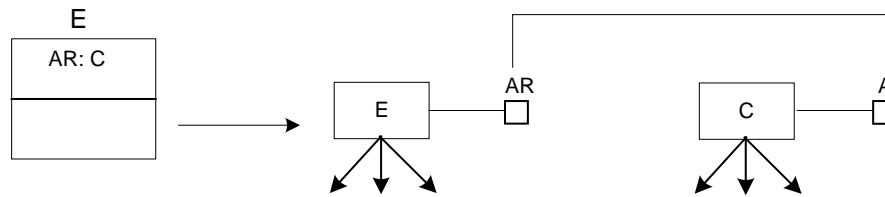
Exemplo:**5.1.7. Regra AR – Mapeamento de Atributo de Referência**

Em um BDOO, quando uma classe realiza associações com outras classes, a classe o faz através de atributos de referência, ou seja, o conteúdo do atributo definido na classe que faz a referência é o OID correspondente da classe referenciada. No entanto, para representar o mesmo conceito em XML, o atributo de referência do esquema OO é mapeado para um atributo do elemento complexo ou tipo complexo correspondente à classe ou estrutura complexa, do tipo IDREF, e ainda, na classe referenciada, deve ser definido um atributo do tipo ID, caso não exista.

Definição: Um *atributo de referência* de nome *AR*, pertencente à classe ou estrutura complexa *E*, que faz referência a uma classe de nome *C* gera:

- Um atributo de referência de nome *AR* do tipo *IDREF* pertencente ao elemento complexo ou tipo complexo correspondente a *E* no esquema XML; e
- Um atributo identificador de nome AI^4 do tipo *ID* pertencente ao tipo complexo correspondente à *C*, se *C* ainda não possui um atributo tipo *ID*;

⁴ *AI* deve ser definido pelo usuário no esquema XSD

Exemplo:

5.1.8. Regra RH1 – Mapeamento de Relacionamento de Herança Simples

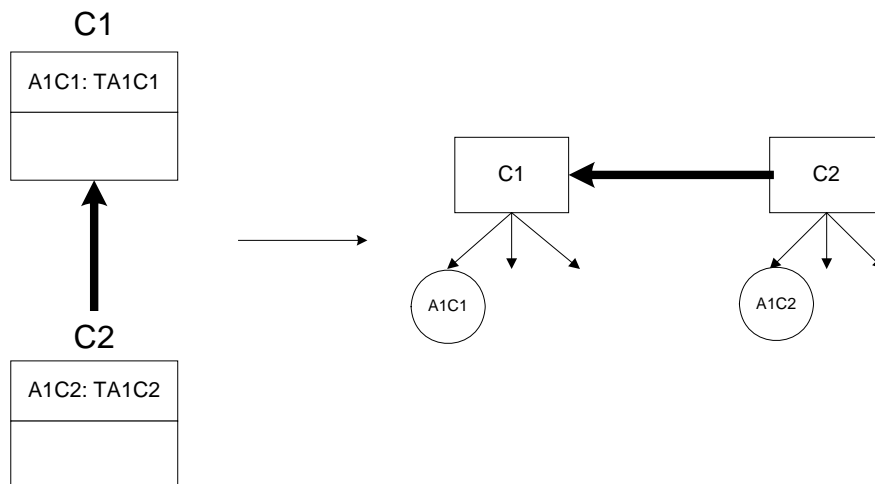
A herança simples pode ser mapeada de três formas distintas, dependendo das características presentes nesta herança. Se a herança consiste em somente redefinir algum atributo da superclasse, é estabelecida uma restrição no tipo complexo correspondente à classe especializada. Se a herança consiste em somente estender a superclasse adicionando novos atributos, define-se uma extensão no tipo complexo correspondente a classe especializada. E caso a herança redefina atributos da superclasse e também estende a mesma adicionando novos atributos, o mapeamento é realizado em dois passos: (i) define-se um novo tipo complexo auxiliar que faz uma restrição sobre a superclasse e; (ii) define-se uma extensão no tipo complexo correspondente a classe especializada sobre o tipo complexo auxiliar.

Definição: Um *relacionamento de herança* existente entre duas classes de nomes $C1$ e $C2$, respectivamente, onde $C2$ é uma especialização de $C1$:

- gera uma restrição, do tipo *restriction*, sobre o tipo complexo correspondente a $C1$ no tipo complexo correspondente a $C2$, se na herança entre $C1$ e $C2$ existe somente redefinição de atributo; ou
- gera uma extensão, tipo *extension*, sobre o tipo complexo correspondente a $C1$ no tipo complexo correspondente a $C2$, se na herança entre $C1$ e $C2$ não existe redefinição de atributo; ou
- gera-se:
 - Um tipo complexo $C1_{aux}$ que faz uma restrição, *tipo restriction*, sobre o tipo complexo correspondente a $C1$; e
 - Uma extensão, tipo *extension*, sobre o tipo complexo $C1_{aux}$ no tipo complexo correspondente a $C2$

, se a herança entre $C1$ e $C2$ existe redefinição de atributo e atributos novos.

Exemplo:



Este exemplo possui uma superclasse chamada de $C1$ e uma classe $C2$ que é uma especialização de $C1$. $C1$ possui um atributo de tipo primitivo e $C2$ estende $C1$ adicionando o atributo $A1C2$ à classe.

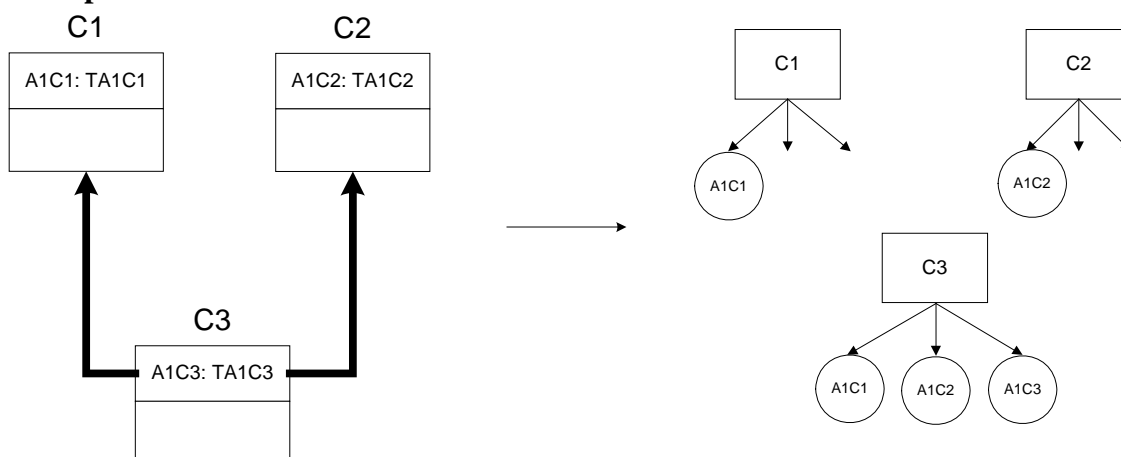
5.1.9. Regra RH2 – Mapeamento de Herança Múltipla

Quando uma classe herda propriedades de mais de uma classe define-se uma herança múltipla. Este conceito, para ser mapeado em uma estrutura XML equivalente, é feito de forma explícita, não tendo nenhum recurso pronto em XML que equivale ao definido numa herança múltipla existente no esquema de BDOO. Portanto, é adicionado um conjunto de estruturas XML equivalentes aos atributos das superclasses, desde que estes não tenham sido redefinidos na classe especializada. Sendo assim, a classe especializada terá as estruturas XML correspondentes aos atributos redefinidos ou estabelecidos nela e também as estruturas XML correspondentes aos atributos das superclasses herdadas.

Definição: Uma hierarquia de herança onde uma classe de nome C_n herda as propriedades de duas ou mais classes de nomes C_1, C_2, \dots, C_m que possuem respectivamente, conjuntos de atributos $\{A_{1C1}:T_{1C1}, A_{2C1}:T_{2C1}, \dots, A_{iC1}:T_{iC1}\}, \dots, \{A_{1Cm}:T_{1Cm}, A_{2Cm}:T_{2Cm}, \dots, A_{iCm}:T_{iCm}\}$, onde cada elemento $A_x:T_x$ destes conjuntos indica, respectivamente, o nome e o tipo do atributo, gera:

- A inclusão em C_n das estruturas XML geradas pelas regras de mapeamento para cada domínio $T_i \in \{T_{1C1}, T_{2C1}, \dots, T_{iC1}, \dots, T_{1Cm}, T_{2Cm}, \dots, T_{iCm}\}$ de um atributo $A_i \in \{A_{1C1}, A_{2C1}, \dots, A_{iC1}, \dots, A_{1Cm}, A_{2Cm}, \dots, A_{iCm}\}$, se A_i não está redefinido em C_n .

Exemplo:



Quando a herança se dá através de extensão, se o tipo complexo correspondente à superclasse possuir um atributo ID, não pode ser declarado outro atributo de mesmo tipo na classe especializada.

5.2. Algoritmo de Mapeamento

Uma vez definidas as regras de mapeamento, é necessário estabelecer uma abordagem de aplicação delas sobre um esquema de um BDOO para obter um esquema XML correspondente.

Um algoritmo de 3 etapas, que define a seqüência de passos a serem aplicados sobre o esquema do BDOO, é apresentado a seguir:

1- Geração dos tipos complexos correspondentes às classes do esquema BDOO:

Para cada classe *C* definida no esquema do BDOO faça

 Se *C* é abstrata então

 Aplica Regra CA

 Senão

 Aplica Regra CC

Para cada classe *C* definida no esquema do BDOO faça

 Para cada atributo *A* especificado em *C* faça

 Caso *A* seja:

- primitivo: Aplica Regra AP;
- de referência: Aplica Regra AR;
- tipo tupla: Aplica Regra ETT;
- tipo lista: Aplica Regra ETL;
- tipo conjunto: Aplica Regra ETC;

2- Geração dos relacionamentos de herança entre as classes:

Para cada classe *C* que seja uma subclasse no esquema do BDOO faça:

 Se *C* possui herança múltipla então

 Aplica Regra RH2;

Senão

Aplica Regra RH1;

3- Geração do esquema XML a partir das definições de tipos:

Especifique um elemento complexo com indicador de ordem sequence de nome ESQUEMA_BDOO;

Para cada tipo complexo definido correspondente a uma classe do esquema do BDOO faça

Especifique um elemento, do tipo definido, filho do elemento complexo ESQUEMA_BDOO, com indicador de ocorrência `minOccurs="0"` `maxOccurs="unbounded"`;

As três etapas que formam este algoritmo visam ordenar os passos necessários para geração do esquema XML correspondente ao esquema do BDOO. Num primeiro momento, são gerados os tipos complexos, com atributos e sub-elementos, que correspondem aos atributos definidos na classe. Estes tipos definem as estruturas correspondentes às das classes do esquema do BDOO.

Uma vez definidos estes tipos complexos, a etapa seguinte tem por objetivo mapear os relacionamentos de herança existentes no esquema do BDOO, aplicando assim a regra correspondente sobre os tipos de heranças encontradas.

Por fim, tendo os tipos definidos com as heranças também mapeadas, a terceira etapa define um elemento complexo que corresponderá ao esquema do BDOO como um todo. Este elemento complexo contém um sub-elemento definido para cada tipo complexo definido pelas etapas anteriores. Estes tipos complexos correspondem a classes concretas do esquema do BDOO, com indicadores de ocorrências *minOccurs* e *maxOccurs* definidos para "0" e "unbounded", respectivamente, com objetivo de permitir que uma instância do XML tenha um número qualquer de instâncias para cada sub-elemento correspondente a uma classe.

6. Estudo de Caso

Com objetivo de validar a abordagem de mapeamento BDOO→XML proposta, um estudo de caso é apresentado neste capítulo. Para este estudo de caso é considerado o domínio de uma universidade. O esquema de um BDOO é a entrada para aplicação da abordagem e através da execução dos passos estabelecidos no algoritmo definido na seção 5.2 é obtido um esquema XML correspondente.

6.1. Requisitos do domínio

Uma universidade possui *Cursos de Graduação* lotados em *Departamentos*. Cada Curso possui um conjunto de *Disciplinas* e estas *Disciplinas* podem pertencer a vários *Cursos*. Cada *Curso* mantém um histórico no BD, sobre seu conjunto de *Alunos* sendo divididos então em *Ex_alunos* ou *Alunos_em_curso*. Também é mantido um histórico das *Disciplinas* cursadas por cada *Aluno*. Os *Professores* lecionam várias *Disciplinas* e estão lotados em um *Departamento*.

A Figura 6-1 apresenta um esquema BDOO para este domínio.

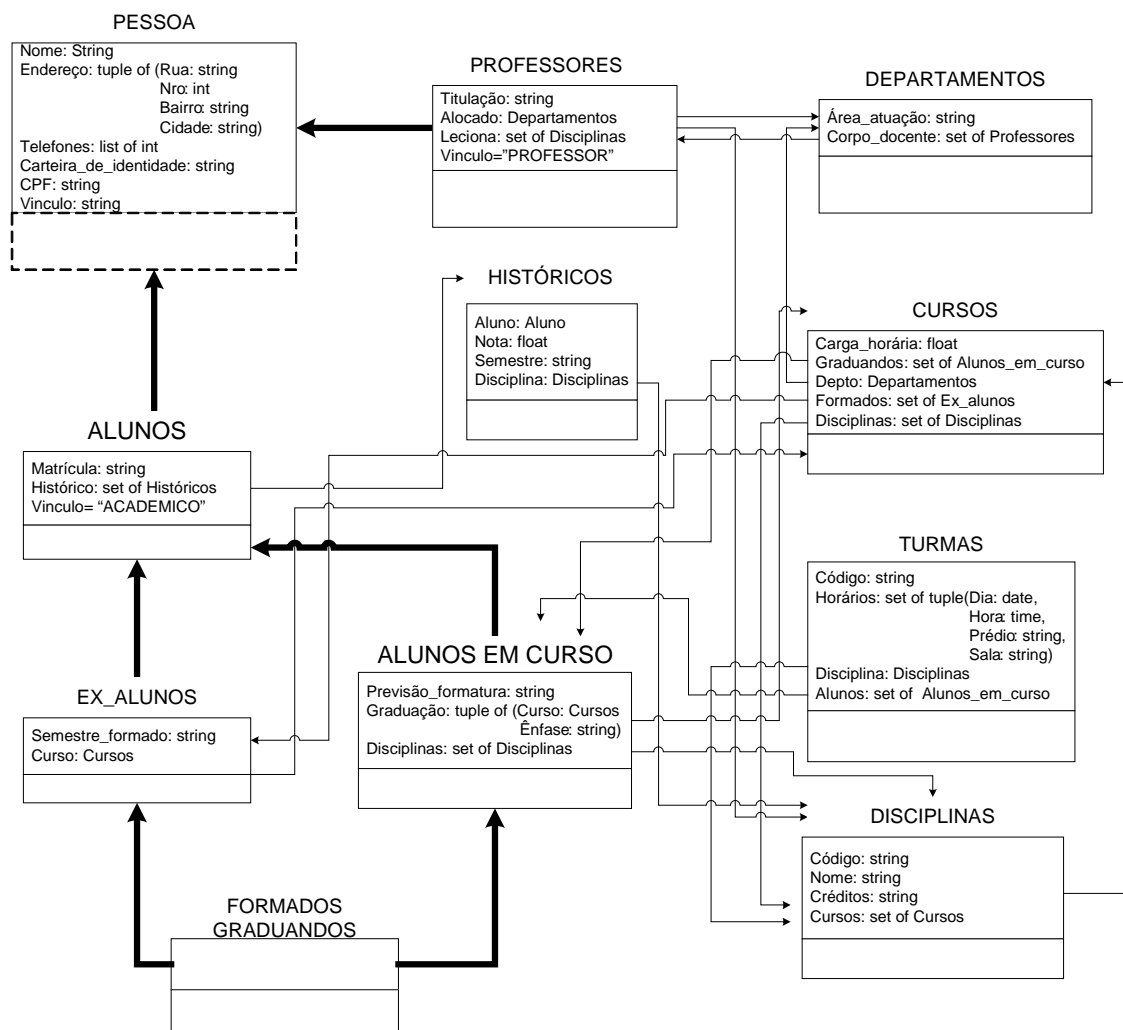


Figura 6-1. Esquema do BDOO da realidade Universitária.

6.2. Aplicação do algoritmo de mapeamento

Uma vez apresentado o esquema do BDOO, é necessário aplicar os passos definidos no algoritmo de mapeamento para obter o esquema XML.

Num primeiro momento, são aplicados os passos definidos na *geração dos tipos complexos correspondentes às classes do esquema do BDOO*. O resultado obtido é um conjunto de definições de tipos complexos contendo atributos e sub-elementos correspondentes aos atributos definidos na classe no esquema XML.

Para obter tais tipos complexos, os passos estabelecem que cada classe seja classificada em concreta ou abstrata e aplicada a regra correspondente ao seu tipo. No esquema do BDOO apresentado na Figura 6-1, somente a classe *Pessoa* é classificada

como abstrata. Depois da classificação da classe, cada atributo dela é também, classificado quanto ao tipo de conteúdo armazenado (primitivo, de referência à outra classe, tupla, lista ou então conjunto). De acordo com a classificação é aplicada a regra de mapeamento correspondente podendo esta aplicação ocorrer de forma recursiva, pois as regras classificam recursivamente suas estruturas para aplicação da regra correspondente. Por exemplo, o atributo *Nome* da classe *Pessoa* é mapeada para um elemento simples (regra TP), já o atributo *Endereço*, como é uma estrutura complexa tipo tupla, foi mapeado para um elemento complexo que possui atributos ou sub-elementos correspondentes aos que se encontram na tupla (regra ETT) e as regras para gerar estes sub-elementos ou atributos do elemento complexo é aplicado de acordo com a classificação do tipo do atributo, de forma recursiva.

Após a geração dos tipos complexos correspondentes às classes, se devem aplicar os passos definidos na *geração dos relacionamentos de herança entre as classes*. No final deste processo é obtido um conjunto de tipos complexos já definidos pelos passos do grupo1, com os tipos complexos correspondentes às subclasses, alterados e também podem surgir algumas definições de tipos complexos auxiliares, como por exemplo, o tipo *Professores_aux* para representar os relacionamentos de herança existente entre as classes. A herança entre as classes é classificada em simples ou múltipla. Uma herança simples é mapeada para uma extensão, restrição ou ambas sobre o tipo correspondente a superclasse, como pode ser visualizado na Figura 6-3 (regra RH1), alguns tipos como *Alunos* e *Professores* herdam da classe *Pessoa* são exemplos deste tipo de herança. Já na herança múltipla, devido a características da linguagem XML Schema, é mapeado de forma diferenciada, devido a restrições da linguagem XML Schema, então leva-se em consideração o que estabelece o conceito de herança entre duas ou mais classes, porque não é permitida a extensão ou restrição de mais de um tipo pela linguagem XML Schema por um elemento ou tipo qualquer. A Classe *Formados_graduandos* utiliza a regra de herança múltipla (regra RH2) para gerar o conceito de herança múltipla no XML.

Por fim, tendo a definição dos tipos complexos correspondentes às classes do esquema do BDOO e os relacionamentos existentes entre elas, aplicam-se os passos definidos na *geração do esquema XML a partir das definições de tipos*. Primeiramente é definido um elemento complexo de nome *ESQUEMA_BDOO* com indicador de

ordem *sequence*. Dentro deste elemento são definidos um elemento, com indicadores de ocorrência *minOccurs* = "0" e *maxOccurs*= "*unbounded*", para cada tipo complexo definido, exceto para os tipos auxiliares e os abstratos, como por exemplo, o tipo *Professores_aux* e *Pessoa*, respectivamente. Com isso é obtido um esquema XML que possui uma estrutura correspondente à definida no esquema do BDOO. Contendo definições de elementos complexos correspondentes às classes concretas definidas no esquema do BDOO.

A Figura 6-3 apresenta o esquema XML correspondente ao esquema BDOO da Figura 6-1, resultante da aplicação da abordagem de mapeamento proposta. A Figura 6-2 apresenta o mesmo esquema XML através da representação gráfica definida. Assim, um conjunto qualquer de dados armazenados no BDOO que segue o esquema considerado pode ser apresentado num documento XML validado pelo esquema XML obtido.

Considerando a existência de instâncias no BDOO, a Figura 6-4 apresenta um conjunto de dados no BDOO seguindo o esquema da Figura 6-1 e a Figura 6-5 mostra o documento XML validado pelo esquema XML gerado para estes dados.

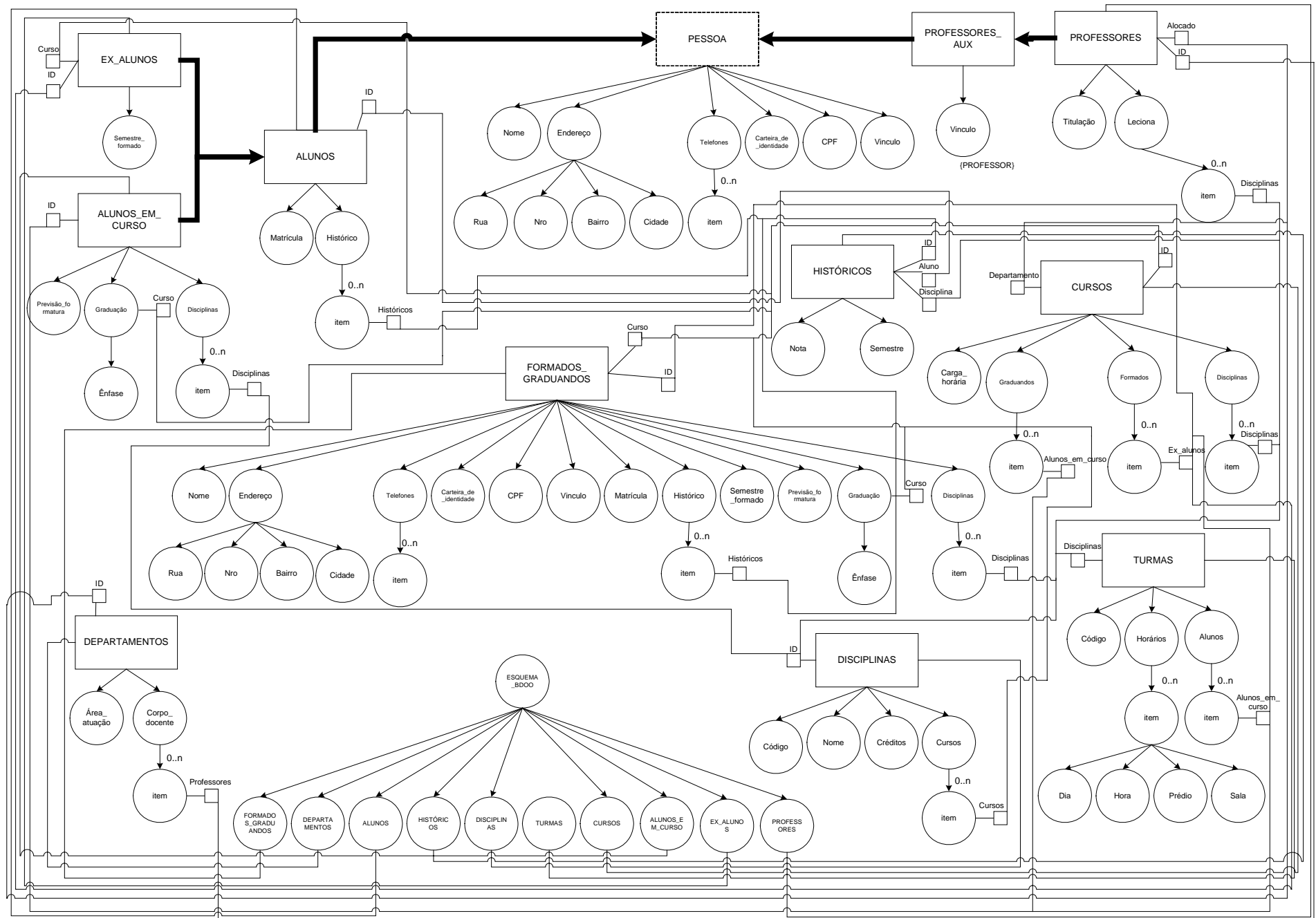


Figura 6-2. Representação gráfica do XML Schema da realidade universitária.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
  <xs:complexType name="PESSOA" abstract="true">
    <xs:sequence>
      <xs:element name="Nome" type="xs:string"/>
      <xs:element name="Endereço">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Rua" type="xs:string"/>
            <xs:element name="Nro" type="xs:int"/>
            <xs:element name="Bairro" type="xs:string"/>
            <xs:element name="Cidade" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Telefones">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="item" type="xs:int" minOccurs="0"
              maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Carteira_de_identidade" type="xs:string"/>
      <xs:element name="CPF" type="xs:string"/>
      <xs:element name="Vinculo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PROFESSORES_AUX">
    <xs:complexContent>
      <xs:restriction base="PESSOA">
        <xs:sequence>
          <xs:element name="Nome" type="xs:string"/>
          <xs:element name="Endereço">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Rua" type="xs:string"/>
                <xs:element name="Nro" type="xs:int"/>
                <xs:element name="Bairro" type="xs:string"/>
                <xs:element name="Cidade" type="xs:string"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Telefones">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="item" type="xs:int" minOccurs="0"
                  maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Carteira_de_identidade" type="xs:string"/>
          <xs:element name="CPF" type="xs:string"/>
          <xs:element name="Vinculo" type="xs:string" fixed="PROFESSOR"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

```

```

<xs:complexType name="PROFESSORES">
  <xs:complexContent>
    <xs:extension base="PROFESSORES_AUX">
      <xs:sequence>
        <xs:element name="Titulação" type="xs:string"/>
        <xs:element name="Leciona">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="Disciplinas" type="xs:IDREF"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:attribute name="Alocado" type="xs:IDREF"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

<xs:complexType name="TURMAS">
  <xs:sequence>
    <xs:element name="Código" type="xs:string"/>
    <xs:element name="Horários">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Dia" type="xs:date"/>
                <xs:element name="Hora" type="xs:time"/>
                <xs:element name="Prédio" type="xs:string"/>
                <xs:element name="Sala" type="xs:string"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Alunos">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Alunos_em_curso" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="Disciplinas" type="xs:IDREF"/>
  </xs:complexType>

<xs:complexType name="DEPARTAMENTOS">
  <xs:sequence>

```

```

<xs:element name="Área_atuação" type="xs:string"/>
<xs:element name="Corpo_docente">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="Professores" type="xs:IDREF"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="CURSOS">
  <xs:sequence>
    <xs:element name="Carga_horária" type="xs:float"/>
    <xs:element name="Graduandos">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Alunos_em_curso" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Formados">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Ex_alunos" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Disciplinas">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Disciplinas" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="Departamento" type="xs:IDREF"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="HISTÓRICOS">
  <xs:sequence>
    <xs:element name="Nota" type="xs:float"/>
    <xs:element name="Semestre" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

```

```

</xs:sequence>
<xs:attribute name="Aluno" type="xs:IDREF"/>
<xs:attribute name="Disciplina" type="xs:IDREF"/>
<xs:attribute name="ID" type="xs:ID"/>
</xs:complexType>

<xs:complexType name="DISCIPLINAS">
  <xs:sequence>
    <xs:element name="Código" type="xs:string"/>
    <xs:element name="Nome" type="xs:string"/>
    <xs:element name="Créditos" type="xs:string"/>
    <xs:element name="Cursos">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Cursos" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:ID"/>
</xs:complexType>

<xs:complexType name="ALUNOS_AUX">
  <xs:complexContent>
    <xs:restriction base="PESSOA">
      <xs:sequence>
        <xs:element name="Nome" type="xs:string"/>
        <xs:element name="Endereço">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Rua" type="xs:string"/>
              <xs:element name="Nro" type="xs:int"/>
              <xs:element name="Bairro" type="xs:string"/>
              <xs:element name="Cidade" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Telefones">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="item" type="xs:int" minOccurs="0"
                maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Carteira_de_identidade" type="xs:string"/>
        <xs:element name="CPF" type="xs:string"/>
        <xs:element name="Vinculo" type="xs:string" fixed="ACADÊMICO"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ALUNOS">
  <xs:complexContent>

```

```

<xs:extension base="ALUNOS_AUX">
  <xs:sequence>
    <xs:element name="Matricula" type="xs:string"/>
    <xs:element name="Histórico">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Históricos" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="ALUNOS_EM_CURSO">
<xs:complexContent>
  <xs:extension base="ALUNOS">
    <xs:sequence>
      <xs:element name="Previsão_formatura" type="xs:string"/>
      <xs:element name="Graduação">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Ênfase" type="xs:string"/>
          </xs:sequence>
          <xs:attribute name="Curso" type="xs:IDREF"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Disciplinas">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="Disciplinas" type="xs:IDREF"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:attribute name="ID" type="xs:ID"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="EX_ALUNOS">
<xs:complexContent>
  <xs:extension base="ALUNOS">
    <xs:sequence>
      <xs:element name="Semestre_formado" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="Curso" type="xs:IDREF"/>
    <xs:attribute name="ID" type="xs:ID"/>
  </xs:extension>
</xs:complexContent>

```



```

</xs:complexType>
<xs:complexType name="FORMADOS_GRADUANDOS">
  <xs:sequence>
    <xs:element name="Nome" type="xs:string"/>
    <xs:element name="Endereço">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Rua" type="xs:string"/>
          <xs:element name="Nro" type="xs:int"/>
          <xs:element name="Bairro" type="xs:string"/>
          <xs:element name="Cidade" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Telefones">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" type="xs:int" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Carteira_de_identidade" type="xs:string"/>
    <xs:element name="CPF" type="xs:string"/>
    <xs:element name="Vinculo" type="xs:string"/>
    <xs:element name="Matricula" type="xs:string"/>
    <xs:element name="Histórico">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Históricos" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Semestre_formado" type="xs:string"/>
    <xs:element name="Previsão_formatura" type="xs:string"/>
    <xs:element name="Graduação">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Ênfase" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="Curso" type="xs:IDREF"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Disciplinas">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="Disciplinas" type="xs:IDREF"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

</xs:sequence>
<xs:attribute name="Curso" type="xs:IDREF"/>
<xs:attribute name="ID" type="xs:ID"/>
</xs:complexType>

<xs:element name="ESQUEMA_BDOO">
<xs:complexType>
<xs:sequence>
<xs:element name="PROFESSORES" type="PROFESSORES" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="TURMAS" type="TURMAS" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="DEPARTAMENTOS" type="DEPARTAMENTOS"
minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="CURSOS" type="CURSOS" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="HISTÓRICOS" type="HISTÓRICOS" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="DISCIPLINAS" type="DISCIPLINAS" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="ALUNOS" type="ALUNOS" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="ALUNOS_EM_CURSO" type="ALUNOS_EM_CURSO"
minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="EX_ALUNOS" type="EX_ALUNOS" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="FORMADOS_GRADUANDOS" type="FORMADOS_GRADUANDOS"
minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

Figura 6-3. Esquema XML da realidade universitária.

```

OID=P1 ⇒ PROFESSORES (“Jose da Silva”, “Rua das árvores”, 123, “Jardim das
flores”, “Fpolis”, (0483316667, 0455356738), “1234567890”, “0090988745”,
“Mestre”, DE1, (DI1))
OID=T1 ⇒ TURMAS (“T123”, ((“quarta”, 8:00, “Prédio 1”, “1”), (“sexta”, 9:00,
“Prédio 1”, “1”)), DI1, (A2))
OID=DE1 ⇒ DEPARTAMENTOS(“Exatas”, (P1))
OID=C1 ⇒ CURSOS(3000, (A2),DE1, (DI1))
OID=HI1 ⇒ HISTORICOS (A2, 8, “1 semestre”, DI1)
OID=DI1 ⇒ DISCIPLINAS(123, “Ética”, “3”, (C1))
OID=A2 ⇒ ALUNOS_EM_CURSO(“Roberto”, “Rua 3”, 3, “Centro”, “Fpolis”,
(0483333337, 0455311111), “1211887777”, “0122999745”, “Estudante”, “133”,
(HI1), “2009”,(C1, “Processamento de Imagens”), (DI1))

```

Figura 6-4. Dados do BDOO

```

<?xml version="1.0" encoding="UTF-8"?>
<ESQUEMA_BDOO>
<PROFESSORES Alocado="DE1" ID="P1">
  <Nome>Jose da Silva</Nome>
  <Endereço>
    <Rua>Rua das árvores</Rua>
    <Nro>123</Nro>
    <Bairro>Jardim das flores</Bairro>
    <Cidade>Fpolis</Cidade>
  </Endereço>
  <Telefones>
    <item>0483316667</item>
    <item>0455356738</item>
  </Telefones>
  <Carteira_de_identidade>1234567890</Carteira_de_identidade>
  <CPF>0090988745</CPF>
  <Vinculo></Vinculo>
  <Titulação>Mestre</Titulação>
  <Leciona>
    <item Disciplinas="DI1"/>
  </Leciona>
</PROFESSORES>
<TURMAS Disciplinas="DI1">
  <Código>T123</Código>
  <Horários>
    <item>
      <Dia>quarta</Dia>
      <Hora> 8:00:00</Hora>
      <Prédio>Prédio 1</Prédio>
      <Sala>1</Sala>
    </item>
    <item>
      <Dia>sexta</Dia>
      <Hora>9:00:00</Hora>
      <Prédio>Prédio 1</Prédio>
      <Sala>1</Sala>
    </item>
  </Horários>
  <Alunos>
    <item Alunos_em_curso="A2"/>
  </Alunos>
</TURMAS>
<DEPARTAMENTOS ID="DE1">
  <Área_atuação> Exatas</Área_atuação>
  <Corpo_docente>
    <item Professores="P1"/>
  </Corpo_docente>
</DEPARTAMENTOS>
<CURSOS Departamento="DE1" ID="C1">
  <Carga_horária>3000</Carga_horária>
  <Graduandos>
    <item Alunos_em_curso="A2"/>
  </Graduandos>
  <Formados></Formados>
  <Disciplinas>
    <item Disciplinas="DI1"/>
  </Disciplinas>

```

```

</CURSOS>
<HISTÓRICOS Aluno="A2" Disciplina="DI1" ID="HI1">
  <Nota>8</Nota>
  <Semestre>1 semestre</Semestre>
</HISTÓRICOS>
<DISCIPLINAS ID="DI1">
  <Código>123</Código>
  <Nome>Ética</Nome>
  <Créditos>3</Créditos>
  <Cursos>
    <item Cursos="C1" />
  </Cursos>
</DISCIPLINAS>
<ALUNOS_EM_CURSO ID="A2">
  <Nome>Roberto</Nome>
  <Endereço>
    <Rua>Rua 3</Rua>
    <Nro>3</Nro>
    <Bairro>Centro</Bairro>
    <Cidade>Fpolis</Cidade>
  </Endereço>
  <Telefones>
    <item>0483333337</item>
    <item>0455311111</item>
  </Telefones>
  <Carteira_de_identidade>12118877777</Carteira_de_identidade>
  <CPF>0122999745</CPF>
  <Vinculo>Estudante</Vinculo>
  <Matricula>133</Matricula>
  <Histórico>
    <item Históricos="HI1" />
  </Histórico>
  <Previsão_formatura>2009</Previsão_formatura>
  <Graduação Curso="C1">
    <Ênfase>Processamento de Imagens</Ênfase>
  </Graduação>
  <Disciplinas>
    <item Disciplinas="DI1" />
  </Disciplinas>
</ALUNOS_EM_CURSO>
</ESQUEMA_BDOO>

```

Figura 6-5. Instância XML validado pelo esquema gerado

7. Conclusão

Esta dissertação apresenta uma abordagem de mapeamento de esquemas de BDOO para esquemas XML em XML Schema. A aplicação deste mapeamento permite a apresentação dos dados de um BDOO em XML, considerando que XML é um padrão atual para representação e intercâmbio de dados entre aplicações.

Alguns trabalhos relacionados ao mapeamento envolvendo objetos e XML são encontrados na literatura, como descrito no capítulo 4. Estes trabalhos possuem objetivos específicos que distinguem dos objetivos buscados nesta dissertação, uma vez que tratam do mapeamento no sentido XML→OO. Entretanto, eles se assemelham ao propósito desta dissertação quando se considera o seu objetivo geral, que é a interoperabilidade entre aplicações através de abordagens de mapeamento entre o padrão XML e objetos. Dentre os trabalhos estudados, três apresentam abordagens para o armazenamento de dados XML em BDOOs, e um deles define regras de mapeamento entre esquemas XML e a UML. A Tabela 4 apresenta uma comparação entre as correspondências dos conceitos XML-OO utilizados pelas abordagens estudadas e a proposta desta dissertação.

Com base nas informações contidas nesta tabela, é possível verificar que além de nenhum deles possuir um objetivo específico idêntico ao proposto, a abordagem desta dissertação é mais completa, pois define correspondências para todos os tipos de estrutura encontrados em um esquema de BDOO. Portanto, a principal contribuição desta dissertação é a definição de uma abordagem detalhada de geração de esquemas XML a partir de um esquema de BDOO.

Tabela 4. Comparação entre as correspondências estabelecidas entre conceitos de BDOO e XML Schema definidos pelos trabalhos relacionados e a abordagem proposta.

BDOO	XML				
	Han, Wook-Shin	Renner, Andreas	Chung, Tae-Sun	Salim, Flora Dilys	Abordagem Proposta
Objetivo Específico	XML Schema-BDOO/OR	XML Schema->Java	DTD->BDOO	XML Schema->UML	BDOO->XML Schema
Classe	elemento	tipos de dados	elemento global	elemento	tipo complexo
Tipo Abstrato	-	-	-	-	tipo complexo abstrato
Atributo Primitivo	elemento/atributo	sub-elemento	elementos filhos com grau um	elemento/atributo	sub-elemento ou atributo
Estrutura Complexa	elemento/atributo			-	sub-elemento complexo
Referências	-	-	-	-	atributos do tipo ID e IDREF
Herança Simples	-	-	diversas possibilidades de sub-classes são geradas, levando em conta a análise de cardinalidade e do construtor <i>choice</i>	tipo complexo derivado por extensão	extensão e/ou restrição de tipos
Herança Múltipla	-	-	-	-	definição de tipo complexo contendo as características herdadas das superclasses

Alguns trabalhos futuros podem ser desenvolvidos para complementar e aprimorar o resultado obtido com esta dissertação. Dentre eles, pode-se citar:

- *Consideração das restrições de integridade definidas no esquema do BDOO*: BDs definem restrições de integridade para garantir uma melhor consistência dos seus dados. Quando se pensa num mapeamento bidirecional de dados, onde eles possam tanto ser mapeados no sentido BDOO→XML quanto no sentido XML→BDOO, é necessário que estes dados mapeados estejam sempre corretos, sendo assim importante o mapeamento das restrições definidas no esquema do BDOO para o esquema XML correspondente;
- *Estudo de abordagens para simplificar o esquema XML resultante*: para aprimorar o resultado obtido, é relevante, analisar e considerar possíveis simplificações nas estruturas XML obtidas como resultado do mapeamento, de modo a obter um esquema XML mais reduzido e legível;
- *Definição de uma abordagem eficiente para representar a herança múltipla encontrada nos BDOOs em XML*: a forma utilizada para representar a herança múltipla entre classes se apresentou complexa e com redundâncias, sendo relevante buscar uma forma mais eficiente de representação XML que elimine estas redundâncias;
- *Desenvolvimento de uma ferramenta de apoio ao processo de mapeamento*: a implementação de uma ferramenta computacional que permita modelar ou importar esquemas de BDOOs e gere esquemas correspondentes em XML é importante para validar e tornar aplicável a abordagem proposta;
- *Definição de um processo de mapeamento de instâncias de dados*: este processo auxiliaria na extração de dados de uma instância do BDOO para armazenamento no esquema correspondente.

Apesar das extensões sugeridas anteriormente, o objetivo pretendido por este trabalho foi alcançado, visto que ele trata o mapeamento de estruturas e relacionamentos definidos num esquema de BDOO e se diferencia de outros trabalhos por sua abrangência. Considerando a necessidade atual de compartilhamento de dados entre

aplicações heterogêneas, este trabalho é uma contribuição para um processo de interoperabilidade entre BDOOs através de XML.

Bibliografia

- [BER93] BERTINO, E.; LORENZO M.. Object-Oriented Database Systems: Concepts and Architectures. Addison-Wesley Publishers Ltd., 1993. 1º edição.
- [BOO99] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML Guia do Usuário. Rio de Janeiro: Editora Campus, 1999. 1º edição.
- [BRA04] BRADLEY, N.. The XML Companion. Harlow: Addison-Wesley Ltd., 2004.
- [CHU01] CHUNG, Tae_Sun; et al. Extracting Object-Oriented database Schemas from XML DTDs Using Inheritance. EC-Web 2001 Munich, Germany, September 4-6.
- [GAI97] GAIDZINSKI, Alexandre. Uma Metodologia para projeto Lógico de Banco de Dados Orientado a Objetos. Florianópolis: Dissertação de mestrado – UFSC. 1997.
- [GUE04] GUEDES, Guilleanes T. A.. UML uma Abordagem Prática. São Paulo: Novatec Editora Ltda, 2004. 1º edição.
- [HAN03] HAN, Wook-Shin, et al; An XML Storage System for Object-Oriented/Object-Relational DBMSs. 2003.
- [HEU98] HEUSER, Carlos A. Projeto de Banco de Dados. Porto Alegre: Editora Sagra Luzzatto, 1998. 1º edição.
- [KIM90] KIM, W. Introduction to Object-Oriented databases. The MIT Press, 1990.

- [MAR01] MARTIN, Didier; et al. Professional XML. Rio de Janeiro: Editora Ciência Moderna Ltda, 2001. 1^o. edição.
- [MAU05] Mauricio, C.R.M.; Peres, F.F.F.; Mello, R.S. Modelo XML Schema: Características e Representação Gráfica. Relatório de Pesquisa do Grupo de Banco de Dados, Número 1. UFSC. Agosto de 2005.
- [MEL05] Mello, R.S.; Heuser, C.A. BInXS: A Process for Integration of XML Schemata. In: 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005), Porto, Portugal, June 13-17, 2005. Proceedings. Lecture Notes in Computer Science, v.3520, Springer-Verlag, 2005. p.151-166.
- [RAY03] RAY, Erik t. Learning XML. USA: O'Reilly & Associates, 2003. 2o. edição.
- [REN01] RENNER, Andreas. XML Data and Object database: The Perfect Couple?; Proceedings of the 17th International Conference on Data Engineering, Washington, DC, USA, p. 143-148, 2001.
- [PRE03] PRESMANN, Roger S. Software Engineering: a Practitioner's Approach. 5. ed.2003. McGraw-Hill.
- [SAL04] SALIM, Flora Dilys; et al. Graphical Representation of XML Schema. Advanced Web Technologies and Applications: 6th Asia-Pacific Web Conference, APWeb 2004, Hangzhou, China, p. 234-245, Abril, 2004.
- [SHA99] SHANMUGASUNDARAM, Jayavel; et al. ; et al. Relational Databases for Query XML Documents: Limitations and Opportunities. In proceedings of Conference on Very Large Data Bases, 1999.
- [SIL99] SILBERSCHATZ, Abraham; Korth, Henry F.; Sudarshan, S.. Sistema de Banco de Dados. Makron Books, 1999. 3^a edição.

- [YER04] YERGEAU, François, et all; Extensible Markup Language (XML) 1.0 (Third Edition) – W3C Recommendation 4th February 2004. disponível em <http://www.w3.org/TR/2004/REC-xml-20040204/>
- [WAT04] WATSON, Richard T.; Data Management – Banco de dados e Organizações Rio de Janeiro :LTC, ano. 3^o edição.
- [W3C04] W3C Recommendation. *XML Schema Part 0: Primer Second Edition*. 2004
XML Schema Part 1: Structures Second Edition
XML Schema Part 2: Datatypes Second Edition.
- [XPA99] W3C Recommendation. *XML Path Language Version 1.0*. 1999

ANEXO 1