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

Cláudio Roberto Marquetto Mauricio

Uma Proposta de Mapeamento do Modelo XML Schema para o Modelo Relacional

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

Ronaldo dos Santos Mello
Orientador

FLORIANÓPOLIS, AGOSTO DE 2005

Uma Proposta de Mapeamento do Modelo XML Schema para o Modelo Relacional

Cláudio Roberto Marquetto Mauricio

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 Wazlawick, Dr.

Banca Examinadora

_____
Ronaldo dos Santos Mello, Dr.

_____
Frank Augusto Siqueira, Dr.

_____
João Bosco Mangueira Sobral, Dr.

_____
Nina Krahe Edelweiss, Dra.

AGRADECIMENTOS

- Agradeço primeiramente a Deus, inteligência suprema e causa primária de todas as coisas, sem Ele, nada disto seria possível;
- Ao meu orientador, prof. Ronaldo dos Santos Mello, cujo apoio e motivação foram fundamentais para a conclusão deste trabalho. Em poucos profissionais encontrei tal equilíbrio entre apoio e cobrança. A você Ronaldo, meus sinceros agradecimentos;
- A Fabiana Frata Furlan Peres, companheira de trabalho na Unioeste e de estudo no mestrado, além de grande amiga. Juntos nos estudos multiplicamos a motivação e dividimos as dificuldades. Parabéns e obrigado;
- A UNIOESTE – Universidade Estadual do Oeste do Paraná e ao Centro de Engenharias e Ciências Exatas, por permitirem meu afastamento parcial para cursar o mestrado;
- Aos amigos da UNIOESTE que me ampararam nas mais variadas formas, para que o trabalho pudesse continuar;
- Ao ITAI-Instituto de Tecnologias Aplicadas a Inovação (antigo Instituto de Tecnologia em Automação e Informática), pela constante busca pelo aprimoramento dos profissionais ao seu redor, afinal tudo começou na especialização promovida por este instituto em parceria com a UFSC;
- Aos professores Juan Carlos Sotuyo e Jorge Habib Hanna El Khouri, professores da minha graduação. Pessoas nas quais ainda hoje procuro me espelhar, pois sempre foram e ainda são, algo como mentores de minha vida profissional. O trabalho realizado por vocês, iniciado com a criação do curso de Ciência da Computação e desdobrado hoje em tantas outras iniciativas, permitiu que, não somente eu mas também muitos outros, construíssemos nossas carreiras profissionais, guiados por uma postura ética, honrada e que não teme desafios. Parabéns e obrigado aos dois!
- Aos amigos pessoais que sempre estiveram na torcida. São muitos e para não correr o risco de esquecer ninguém agradeço a todos vocês. Obrigado Amigos;

- Ao Frank, esposo da Fabiana, um abraço agradecido. Em muitas viagens ele foi nosso companheiro, confidente, conselheiro, assistente, chofer, e muito mais, resumindo, foi um verdadeiro amigo que sempre fez tudo ao seu alcance para nos ajudar;
- Um agradecimento muito especial à minha família, que em todos os momentos, principalmente naqueles em que a esperança parecia desfalecer, foi recanto para reequilibrio e fonte de coragem. À minha esposa, que com paciência suportou minhas dificuldades e nervosismos, e com ternura me incentivou durante esta caminhada. És a musa inspiradora de meus passos; Às minhas filhas Stela e Sarah, que em muitos momentos se viram privadas da presença e atenção do pai, e mesmo assim mantiveram-se como geradores de alegria e paz em meu coração; Aos meus irmãos Márcia(tia Má), Marcio(tio Didi) e Carlos Renato (tio Nato), minhas cunhadas e cunhado Juliane, Michele, Jankiel, Juliane, Larissa e Paola, meus sobrinhos Regina, Maiara, Júlia, Maria Luiza, Rennan e Isadora, e a minha sogra e amiga Marilene. A todos meu muito obrigado, pois em seus pensamentos sempre esteve a torcida e o carinho por mim. Ao "tio" e amigo Luiz Carlos e ao Diogo, pelas acolhidas em cada viagem para Florianópolis. Graças à sua presteza, pude me dedicar com mais tranqüilidade às minhas atividades; E por fim, aos meus pais Valentim e Alice, sempre preocupados e atenciosos, me incentivaram nesta jornada. Foram eles os primeiros, e mais importantes, professores de minha vida, as lições que deles recebi, possuem o maior valor de todos, pois me ensinaram a ser um homem de bem.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

RESUMO

O uso da XML como padrão para intercâmbio de dados gera a necessidade de um esquema comum a ser seguido pelos sistemas envolvidos. Os mecanismos mais usados para a definição de esquemas XML são a DTD e a XML Schema. Com estas tecnologias, é possível definir a estrutura a ser seguida pelos documentos XML a serem intercambiados, estabelecendo um protocolo de troca de informações independente do mecanismo de armazenamento de dados usado pelos sistemas.

Neste contexto, é necessário que os sistemas comunicantes sejam capazes de transformar seu modelo de dados em XML para o modelo de dados utilizado pelo sistema e vice-versa. O modelo de dados relacional é utilizado por muitos destes sistemas, considerando a grande disponibilidade de Sistemas Gerenciadores de Bancos de Dados (SGBDs) que adotam este modelo. Para isso, estes sistemas devem desenvolver mecanismos para exportar os dados de suas tabelas no formato XML e também para decompor documentos XML e armazená-los no SGBD. Estes mecanismos devem ser genéricos, dinâmicos e eficientes para garantir uma atividade adequada de intercâmbio de dados.

Assim sendo, este trabalho propõe um mecanismo baseado em regras para transformar um esquema de dados XML, definido com o uso de *XML Schema*, para um esquema de dados relacional, que pode ser usado por SGBDs relacionais existentes no mercado. Como contribuição específica, este trabalho realiza uma análise detalhada dos conceitos do modelo *XML Schema*, considerando estes conceitos nas regras para transformação do *XML Schema* em um esquema de dados relacional.

ABSTRACT

The use of XML as a protocol for data interchange brings up the need for a common schema to be used by the involved systems. The most used mechanisms to define XML schemas are DTD and XML Schema. With these technologies, it is possible to define the structure to be followed by the XML documents that will be interchanged, establishing a protocol for information exchange that is independent of the data recording mechanism used by these systems.

Within this context, the comunicating systems need to be able to transform it's data model into XML and vice-versa. The relational data model is used by many of these systems, considering the wide availability of DataBase Management Systems (DBMS) that adopt this model. In order to do this, the systems must develop mechanisms to export the data of their tables into XML format and also decompose XML documents and store them in the DBMS. These mechanisms must be generic, dynamic and efficient to ensure a suitable data interchange.

This work proposes a rule-based mechanism to transform an XML data schema, defined through *XML Schema*, into a relational data schema, which can be used by available relational DBMSs. As a specific contribution, this work provides a detailed analysis of the *XML Schema* concepts, taking these concepts into consideration in the rules that transform a schema in XML Schema into a relational data schema.

1. INTRODUÇÃO

Muitas empresas possuem ambientes de trabalho distribuídos onde usuários precisam trocar informações entre diversos sistemas diferentes. Para facilitar este processo, normalmente define-se um modelo comum para o intercâmbio. XML é, hoje, amplamente utilizado para agilizar e facilitar a troca de informação entre estes sistemas [FRA04]. A flexibilidade e a extensibilidade do XML permitem a criação de protocolos de comunicação de dados entre sistemas, permitindo importar e exportar informações dos mesmos [MUR03].

A motivação inicial da XML era a troca de informações através de documentos eletrônicos. Entretanto, devido à sua simplicidade e fácil portabilidade, logo, as empresas começaram a adotá-lo como modelo para a troca de informações e desenvolvimento de aplicações Web.

O uso da XML como padrão para intercâmbio de dados gera a necessidade de um esquema comum a ser seguido pelos sistemas envolvidos. Os mecanismos mais usados para a definição de esquemas XML são a *DTD (Document Type Definition)* e a *XML Schema (XSD)* [W3C04]. Com estas tecnologias, é possível definir a estrutura a ser seguida pelos documentos XML a serem intercambiados, estabelecendo um protocolo de troca de informações independente do mecanismo de armazenamento de dados usado pelos sistemas [MUR03].

Neste contexto, é necessário que os sistemas comunicantes sejam capazes de transformar seu modelo de dados em XML para o modelo de dados utilizado pelo sistema e vice-versa. O modelo de dados relacional é utilizado por muitos destes sistemas, considerando a grande disponibilidade de Sistemas Gerenciadores de Bancos de Dados (SGBDs) que adotam este modelo. Para isso, estes sistemas devem desenvolver mecanismos para exportar os dados de suas tabelas no formato XML e também para decompor documentos XML e armazená-los no SGBD. Estes mecanismos devem ser genéricos, dinâmicos e eficientes para garantir uma atividade adequada de intercâmbio de dados. [FER00].

Assim sendo, este trabalho tem por objetivo estudar e propor um mecanismo baseado em regras para transformar um esquema de dados XML, definido com o uso de

XML-Schema [W3C04], para um esquema de dados relacional, que pode ser usado por SGBDs relacionais existentes no mercado. Como contribuição específica, este trabalho realiza uma análise detalhada dos conceitos do modelo XML Schema, considerando estes conceitos nas regras para transformação do XML Schema em um esquema de dados relacional. A escolha de XML-Schema se dá pelo fato desta ser a recomendação atual da W3C para representação de esquemas XML e também por apresentar mais recursos para a definição de esquemas XML. Em trabalhos similares nem todos os recursos da linguagem são considerados e por isso este trabalho considera o conjunto completo de conceitos desta recomendação. Já o modelo relacional foi escolhido por ser o modelo mais utilizado por SGBDs comerciais, e pela sua maturidade, alcançada após décadas de pesquisa e desenvolvimento.

O Capítulo 2 contém uma breve revisão sobre conceitos do modelo relacional, sua forma de representação gráfica em tabelas e análise de sistemas gerenciadores de banco de dados comerciais. O capítulo 3 apresenta uma análise detalhada dos conceitos do modelo XML Schema. O Capítulo 4 faz uma análise de trabalhos relacionados. No Capítulo 5, a abordagem de mapeamento do modelo XML Schema para o modelo relacional é apresentada e o Capítulo 6 exemplifica a aplicação desta abordagem em um estudo de caso. Por fim, o Capítulo 7 apresenta a conclusão desta dissertação e trabalhos futuros. O ANEXO 1 [MAU05] apresenta de maneira detalhada os recursos da linguagem XML Schema.

2. XML e XML Schema

Para tratar do mapeamento entre os dois modelos abordados por este trabalho é necessário conhecer a linguagem XML e também a linguagem XML Schema. Neste capítulo são apresentados brevemente os recursos das mesmas. No ANEXO 1 [MAU05] estes conceitos são abordados de maneira aprofundada e complementam o conteúdo apresentado neste capítulo, sendo recomendada a sua leitura na seqüência.

2.1. Visão Inicial

A W3C (*World Wide Web Consortium*) é responsável por recomendações e pela criação de padrões para a rede mundial de computadores. Dentre os padrões da W3C está a *XML* (eXtensible Markup Language), que é uma linguagem de marcação usada para descrição de informações.

XML é uma linguagem flexível e robusta para descrição de dados, pois não possui elementos pré-definidos. XML é considerada uma meta-linguagem, pois não possui elmentos pré-definidos. O usuário ou desenvolvedor define um conjunto de elementos que seja adequado para representação de seus dados no contexto de sua aplicação.

A necessidade de intercâmbio de informações na *Web* de maneira simples foi a motivação inicial para a definição da XML. 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.

Um documento XML é um arquivo texto que contendo marcações e dados que formam uma árvore hierárquica. A Figura 2-1 apresenta um exemplo de documento XML, ou instância XML, com dados a respeito de um automóvel. As marcações XML são chamadas de **elementos**. Um exemplo de elemento no documento XML é `fabricante`. Cada um dos elementos XML é definido por uma *tag inicial* (`<nome>`) e uma *tag final* (`</nome>`). Um elemento pode conter outros elementos, um valor textual, ou ambos [MAR01].

Os elementos XML podem ainda possuir atributos para descrição adicional de informações relacionadas com os mesmos. Para definir atributos usa-se um par nome-valor especificados na *tag* de início do elemento que contém o atributo. O elemento

`automóvel` no documento XML possui um atributo `placa` (`<automóvel placa="AAA-9999">`).

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
 <automóvel placa="AAA-9999">
   <modelo>Corsa Sedan</modelo>
   <cor>Prata</cor>
   <fabricante>
     <razao>General Motor do Brasil</razao>
     <endereço>
       <rua>das montadoras</rua>
       <numero>001</numero>
       <cep>88888-888</cep>
       <cidade>São Caetano</cidade>
     </endereço>
   </fabricante>
  </pessoa>
```

Figura 2-1. Documento XML.

Informações sobre um mesmo contexto podem estar descritas em documentos XML. Estes documentos não necessariamente possuem uma mesma estrutura já que as marcações do documento não são pré-definidas pela linguagem XML, mas sim pelo usuário. Em situações onde estes documentos precisem seguir uma única estrutura, é preciso criar uma maneira de garantir isto. A linguagem *XML Schema* (*eXtensible Markup Language Schema*) pode ser usada como a linguagem de definição de esquemas para documentos XML.

XML Schema é uma linguagem de marcação que usa o padrão XML, e é a recomendação atual da W3C para definição de esquemas de documentos XML [W3C04].

Um esquema definido em *XML Schema* - chamado esquema *XSD (XML Schema Definition)* - define estruturas XML declarando elementos, sua ordem, restrições de conteúdo, atributos, grupos reusáveis, novos tipos de dados, enfim, tudo que for necessário para estabelecer o que uma instância de um documento XML, em conformidade com este esquema, pode ou deve ter.

Um esquema XSD é formado por um elemento raiz chamado *schema*. Este elemento pode ter vários elementos filhos. Toda a estruturação do esquema se encontra dentro deste elemento.

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

- grupo de gerenciamento;

- grupo de componentes de construção do modelo.

Todas as declarações do primeiro grupo devem estar antes de qualquer do segundo grupo [BRA04]. O grupo de gerenciamento possui elementos que são usados para gerenciar a definição do esquema quando este está distribuído em vários arquivos. O grupo de componentes de construção do modelo possui as definições de elementos, atributos, tipos de dados e outras informações para a definição do esquema. A Figura 2-2 apresenta os elementos que compõem estes dois grupos de declarações.

```
<schema>
   <!--GRUPO DE GERENCIAMENTO-->
   <import>…</import>
   <include>…</include>
   <redefine>…</redefine>
   <!—GRUPO DE COMPONENTES DE CONSTRUÇÃO DO MODELO-->
   <notation>…</notation>
   <simpleType>…</simpleType>
   <complexType>…</complexType>
   <attributeGroup>…</attributeGroup>
   <attribute…/>
   <group>…</group>
   <element>…</element>
</schema>
```

Figura 2-2. Grupos de Declarações.

Para este trabalho o grupo de componentes de construção é o mais importante, visto que os dados do domínio de aplicação propriamente ditos são definidos através de seus elementos.

2.2. Recursos da linguagem XML Schema

A linguagem *XML Schema* possui um conjunto de elementos pré-definidos para permitir a definição de esquemas. Estes elementos, por sua vez, possuem conjuntos de atributos necessários para detalhar suas características, tais como nome, seu uso (opcional ou obrigatório), valor *default*, entre outras. A Figura 2-3 mostra um exemplo de esquema XML que faz uso destes recursos da linguagem XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
 <xs:element name="e12" type="xs:float"/>
```

```
  <xs:complexType name="t1" abstract="true">
    <xs:sequence>
      <xs:element name="e20" type="xs:string"/>
      <xs:element name="e21" type="xs:t2"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="t2">
    <xs:sequence>
      <xs:element name="e22" type="xs:string"/>
      <xs:element name="e23" type="xs:int"/>
    </xs:sequence>
    <xs:attribute name="a7" type="xs:IDREF"/>
  </xs:complexType>
<xs:group name="g1">
  <xs:sequence>
   <xs:element name="e18">
    <xs:complexType>
       <xs:attribute name="a4" type="xs:date"/>
       <xs:attribute name="a5">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="x"/>
         <xs:enumeration value="y"/>
         <xs:enumeration value="z"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="ID" type="xs:ID"/>
   </xs:complexType>
  </xs:element>
  <xs:element name="e19" type="xs:string"/>
   </xs:sequence>
</xs:group>
<xs:element name="e1">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="e2">
     <xs:complexType>
      <xs:complexContent>
       <xs:extension base="t1"/>
      </xs:complexContent>
     </xs:complexType>
    </xs:element>
    <xs:element name="e3" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:choice>
       <xs:element name="e10" type="xs:integer"/>
       <xs:element name="e11" type="xs:string" maxOccurs="unbounded"/>
       <xs:element ref="e12"/>
      </xs:choice>
      <xs:attribute name="a1" type="xs:string" use="prohibited"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="e4"  minOccurs="0"/>
    <xs:element name="e5">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="e12"/>
       <xs:element name="e13" type="xs:time" maxOccurs="2"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="e6" minOccurs="0">
     <xs:complexType/>     </xs:element>
    <xs:element name="e7" minOccurs="0">
```

```
     <xs:complexType>
      <xs:all>
       <xs:element name="e14" type="xs:string"/>
       <xs:element name="e15" type="xs:integer"/>
      </xs:all>
     </xs:complexType>
    </xs:element>
    <xs:element name="e8">
     <xs:complexType>
      <xs:attribute name="a2" type="xs:string" use="required"/>
      <xs:attribute name="a3" type="xs:string" use="optional"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="e9" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType mixed="true">
      <xs:choice>
       <xs:element name="e16" type="xs:string"/>
       <xs:element name="e17" type="xs:string"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:group ref="g1" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

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

Conforme mostra a Figura 3-3, os principais elementos de definição da XML Schema são os seguintes:

- **<schema> </schema>**: elemento raiz de qualquer esquema XML. Todas as definições do modelo devem estar contidas dentro deste elemento;

- **<element name="nomeElemento" type="tipoDado" maxOccurs="numMaxOcorrencias"> </element>:** declaração de elementos que irão compor o esquema. Possui vários atributos que o definem, como:
    - name – define o nome do elemento;
    - type – define o tipo do elemento;
    - minOccurs – define o número mínimo de ocorrências;
    - maxOccurs – define o número máximo de ocorrências;
    - abstract – define que o elemento não pode ser instanciado (semelhante ao paradigma Orientado a Objetos);

- **<complexType name="t1" abstract="true"> </complexType>**: definição de tipos de dados complexos. Estes tipos podem ser usados na criação

de elementos ou de novos tipos. Assim como *element*, ele também possui atributos que fazem parte da sua definição;

- **<simpleType> </simpleType>**: definição de tipos de dados simples. Estes tipos podem ser usados na criação de elementos, atributos ou de novos tipos;

- **<union memberTypes="tipoDado"> </union>**: definição de tipos de dados simples que são uniões de outros tipos simples. Os valores admitidos pelo tipo podem ser de qualquer um dos tipos incluídos na união;

- **<list itemType="tipoDado"/>**: definição de tipos de dados simples que são listas de itens do tipo definido pelo atributo *itemType*. Este tipo dever ser simples;
- **<group name="nomeGrupo"> </group>**: este elemento é usado para declarar um grupo de elementos. Quando se define um grupo de elementos é necessário especificar a ordem destes fazendo uso de indicadores de ordem (*sequence, all, choice*);

- **<attributeGroup name="nomeGrupoAtributos"> </attributeGroup>**: este elemento é usado para declarar um grupo nomeado de atributos. Este grupo pode ser utilizado na declaração de outros elementos, com o objetivo de uso ou reuso da declaração do grupo de atributos.

- **<attribute name="nomeAtributo" type="tipoDado" use="prohibited"/>**: definição de atributos de elementos. O seu tipo deve ser simples;

- **<sequence> </sequence>**: indicador de ordem usado na definição de tipos complexos para estabelecer que os elementos presentes em sua declaração devem aparecer nas instâncias na mesma ordem em que foram declarados;

- **<choice> </choice>**: indicador de ordem usado na definição de tipos complexos para estabelecer que se deve escolher um dentre os elementos presentes em sua declaração;

- **<all> </all>**: indicador de ordem usado na definição de tipos complexos para estabelecer que os todos os elementos presentes em sua declaração devem aparecer nas instâncias.

- **<restriction base="string"> </restriction>**: criação de tipo derivado com restrições de conteúdo adicionais. Ele pode ser aplicado a tipos simples e tipos complexos, gerando desta maneira um novo tipo que é um subconjunto do tipo base;

- **<extension base="t1"/>**: criação de tipo derivado com extensão de conteúdo. Ele pode ser aplicado a tipos complexos, gerando desta maneira um novo tipo que herda as características do tipo base, podendo estendê-lo.

- **<complexContent> </complexContent>**: este elemento é usado em conjunto com o elemento *complexType* para indicar conteúdos derivados de tipos complexos;

- **<simpleContent> </simpleContent>**: este elemento é usado em conjunto com o elemento *complexType* para indicar a definição de conteúdo simples derivado de outro tipo simples ou de um tipo complexo de conteúdo simples;

- **<element name="e1" type="anyType"> </element>**: XML Schema permite que se definam elementos do tipo *anyType,* ou seja, elementos que não possuem nenhuma estrutura pré-definida, podendo ser instanciados com um conteúdo qualquer. Para que isso seja possível, o atributo *type* do elemento deve ser do tipo *anyType*;

- **<complexType name="t1" mixed="true"> </complexType>**: esta é uma declaração de tipo de elemento misto. Um elemento misto pode conter sub-

elementos, texto e atributos. Para definir elementos mistos, o atributo *mixed* do tipo complexo do elemento deve ser definido como *true.*

No ANEXO 1, os recursos da linguagem XML Schema, tratados nesta seção, são apresentados de maneira detalhada.

3. MODELO RELACIONAL

O modelo relacional, proposto por E.F. Codd, do laboratório de pesquisa da IBM, usa um conjunto de tabelas para representar dados e os relacionamentos entre estes dados [COD70]. É um modelo de dados que compreende: 1) uma coleção de tipos de dados; 2) uma coleção de operadores ou regras de inferência e; 3) uma coleção de regras de integridade [COD80]. Desde a sua criação, o modelo tem sido usado como base para implementar sistemas gerenciadores de banco de dados [SIL99].

Os principais conceitos presentes no modelo relacional são [SIL99] [ELM02]:

- **Relação**: consiste em um conjunto de dados relacionados entre si. Apresenta o nome destes dados no cabeçalho e as ocorrências destes dados no seu corpo, onde cada linha representa uma relação entre os dados (atributos);
- **Domínio**: conjunto de valores atômicos que podem ser assumidos por um componente de uma relação;
- **Atributo**: define o papel de um domínio em uma relação, armazenando o valor indicado em um determinado domínio. Na prática um atributo é representado por uma coluna da tabela;
- **Tupla**: define um fato do mundo real pelo registro das informações nos atributos que representam o fato. Na prática uma linha da tabela é uma tupla;
- **Chave primária**: atributo ou conjunto de atributos que pelo seu valor identifica unicamente uma tupla em uma relação;
- **Chave estrangeira**: atributo ou conjunto de atributos que estabelece uma equivalência de valor com a chave primária da mesma relação ou de outra relação.

A representação gráfica do modelo relacional [CHE76], muito utilizada em livros didáticos como [ELM02] e [SIL99], é a representação tabular, consistindo de tabelas para as relações. Cada tabela, contém linhas (tuplas) e colunas (atributos). Considera-se nesta dissertação, que as chaves primárias são representadas por um "sublinhado" no atributo e as chaves estrangeiras por um "sobrelinhado" no atributo. A Figura 3-1 ilustra esta representação do modelo, para o domínio de uma Universidade.

Assume-se ainda que, para registrar informações adicionais sobre as tabelas ou atributos, como restrições de integridade, deve-se usar "anexos", que são descrições textuais informais.

Professores

CPF	Nome	Rua	Número	Cidade	Salário	Titulação	DN

Disciplinas

Código	Nome	Créditos

Docência

CPF	Código

Anexo:
- O atributo CPF deve conter uma seqüência de nove dígitos seguidos de um traço, - , e mais dois dígitos conforme este exemplo: 999999999-99

Figura 3-1. Modelo Relacional

Maiores detalhes sobre o modelo relacional podem ser encontrados em [SIL99] e [ELM02].

3.1. Sistemas Gerenciadores de Banco de Dados Relacionais e o Suporte a XML

Algum tempo depois da proposta do modelo relacional surgiram implementações de Sistemas Gerenciadores de Banco de Dados (SGBDs) que o tomam por base. Por se tratar de um modelo com uma base sólida, usando princípios matemáticos da teoria dos conjuntos e da álgebra relacional, após as primeiras versões, os sistemas rapidamente incluíram otimizações para armazenamento, consultas e outras operações associadas. Ao longo de mais de vinte anos de desenvolvimento, estes SGBDs adquiriram maturidade e são considerados bastante confiáveis [AME04].

Com o crescente uso de XML e sistemas de banco de dados, a integração dos mesmos, quanto ao armazenamento, recuperação e atualização, tornou-se uma necessidade [KAP04]. Para atender esta necessidade, estes SGBDs buscaram soluções para armazenamento e publicação de dados XML de maneira a atender os requisitos de seus usuários, permitindo a manutenção do uso de somente um SGBD, evitando a aquisição de outro para suportar XML.

SGBDs comerciais provêem diferentes mecanismos para mapeamento que podem ser automáticos, usando um modelo pré-definido estático, ou manual, onde o usuário deve definir manualmente o mapeamento usando uma linguagem proprietária fornecida pelo desenvolvedor do banco [AME04].

Atualmente existem várias opções de SGBDs no mercado, entre eles destacam-se o IBM DB2, Microsoft SQL Server e Oracle. A seguir são descritos os mecanismos para suporte à XML destes três SGBDs.

3.1.1. IBM DB2 XML Extender

A IBM oferece o *XML Extender*[1] para armazenamento de documentos XML, para isto usa a linguagem *Document Access Definition-DAD*. A linguagem DAD usa dois elementos (*element_node* e *attribute_node*) para descrever a estrutura de documentos XML e um elemento (*RDB_node*) para descrever o mapeamento desta estrutura para o esquema do banco de dados. O elemento *element_node* especifica os elementos, e o elemento *atribute_node* especifica os atributos presentes em um documento XML. Os dois possuem o subelemento *RDB_element* que por sua vez possui os subelementos *table*, *column* e *condition.* Cada um destes elementos especifica respectivamente uma tabela, coluna e relacionamentos do tipo chave primária-chave estrangeira entre tabelas da esquema do banco de dados[HAN03][AME03] .

Nos mapeamentos usando DAD, elementos XML são mapeados para tabelas ou colunas em tabelas, atributos XML são mapeados para colunas e relacionamentos entre elementos XML são mapeados através de relacionamentos com chave primária e chave estrangeira entre tabelas. Os passos necessários para realizar este mapeamento são[HAN03]:

- especificar a estrutura do documento XML usando os elementos *element_node* e *attribute_node;*
- especificar o mapeamento para o esquema do banco de dados usando o elemento RDB_node;
- especificar todos os relacionamentos entre tabelas com chave primária e chave estrangeira no *RDB_node* filho do elemento *element_node* raiz;

- especificar a tabela ou coluna para a qual um elemento ou atributo é mapeado no *RDB_node* dos demais *element_node* e dos *atribute_node*.

```
<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>
</xsd:schema>
```

Figura 3-2. Exemplo de XML Schema. Fonte [HAN03].

A Figura 3-3 é uma definição DAD que especifica o mapeamento da estrutura do documento XML apresentado na Figura 3-2 para o esquema do banco de dados apresentado na Figura 3-4.

[1] IBM DB2 XML Extender. *http://www-4.ibm.com/software/data/db2/extenders/xmlext/docs/v71wrk/english/index.htm*

```
<element_node name="book">
      <RDB_node>
            <table name="book" key="id"/>
            <table name="author"/>
            <condition> book.id=author.bookId </condition>
      </RDB_node>
      <attribute_node name="id">
            <RDB_node>
                  <table name="book"/>
                  <column name="id" type="integer"/>
            </RDB_node>
      </attribute_node>
      <element_node name="title">
            <text_node>
                  <RDB_node>
                        <table name="book"/>
                        <column name="title" type="varchar(100)"/>
                  </RDB_node>
            </text_node>
      </element_node>
      <element_node name="author" multi_occurrence="YES">
            <element_node name="name">
                  <text_node>
                        <RDB_node>
                              <table name="author"/>
                              <column name="name" type="varchar(100)"/>
                        </RDB_node>
                  </text_node>
            </element_node>
            <element_node name="email">
                  <text_node>
                        <RDB_node>
                              <table name="author"/>
                              <column name="email" type="varchar(100)"/>
                        </RDB_node>
                  </text_node>
            </element_node>
      </element_node>
</element_node>
```

Figura 3-3. Um exemplo DAD. Fonte [HAN03]

Table book

name	id	title
type	integer	varchar(100)

Table author

name	bookId	name	Email
type	integer	varchar(100)	varchar(100)

Figura 3-4. Exemplo de um esquema do banco de dados. Fonte: [HAN03].

3.1.2. Microsoft SQL Server[2]

O SGBD Microsoft SQL Server oferece armazenamento de dados XML através da ferramenta *XML Bulk Load*. A linguagem para mapeamento é chamada XML-Data Reduced (XDR) Schema. Trata-se de uma linguagem que faz anotações em um XML Schema usando quatro elementos (*ElementType, AttributeType, element, attribute*) para

especificar a estrutura de um documento XML, dois atributos (*relation* e *field*) e um elemento (*relationship*) para especificar o mapeamento para um esquema do banco de dados [HAN03].

Os elementos *ElementType* e *AttributeType* são usados para declarar elementos e atributos XML e *element* e *attribute* são usados para fazer referências aos *ElementType* e *AttributeType* declarados. Por fim, os atributos *relation* e *field* especificam uma tabela e coluna, e o elemento *relationship* especifica um relacionamento por chave primária e chave estrangeira entre duas tabelas do esquema do banco.

Assim com nos mapeamentos usando DAD, o XDR Schema anotado mapeia elementos e atributos XML para tabelas ou colunas no banco de dados, e relacionamentos entre elementos XML são mapeados através de relacionamentos com chave primária e chave estrangeira entre tabelas. Os passos necessários para realizar este mapeamento são[HAN03]:

- Especificar a estrutura do documento XML usando os elementos *ElementType, AttributeType, element e attribute;*
- Especificar a tabela ou coluna para a qual um elemento ou atributo é mapeado usando os atributos *relation* e *field*.
- Especificar todos os relacionamentos entre tabelas com chave primária e chave estrangeira usando o elemento *relationship*;

A Figura 3-5 mostra um exemplo de um XDR Schema, equivalente ao esquema da Figura 3-3, que especifica o mapeamento da estrutura do documento XML apresentado na Figura 3-2 para o esquema do banco de dados apresentado na Figura 3-4.

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:sql="urn:schemas-
microsoft-com:xml-sql">
   <ElementType name="book" sql:relation="book">
      <element type="title" sql:field="title"/>
      <element type="author" minOccurs="0" maxOccurs="*">
         <sql:relationship key-relation="book" key="id"
            foreign-relation="author" foreign-key="bookId"/>
      </element>
      <attribute type="id" sql:field="id"/>
```

[2] Microsoft Corp.: Microsoft SQL Server 2000 http://www.microsoft.com/sql/default.asp.

```
</ElementType>
<ElementType name="title" dt:type="string"/>
<AttributeType name="id" dt:type="int"/>
<ElementType name="author" sql:relation="author">
   <element type="name" sql:field="name"/>
   <element type="email" sql:field="email"/>
</ElementType>
<ElementType name="name" dt:type="string"/>
<ElementType name="email" dt:type="string"/>
</Schema>
```

Figura 3-5. Exemplo de XDR Schema.

Este SGBD provê também uma outra forma para armazenar documentos XML chamada OpenXML. Esta solução compila documentos XML para uma representação interna usando DOM[3] (*Document Object Model*). A função chamada T-SQL constrói linhas de uma tabela a partir de uma entrada XML [AME03]. De modo geral, esta forma permite a definição de visões relacionais a partir de dados XML[KAP04]. Um exemplo da sintaxe pode ser visto na Figura 3-6.

```
Select * from OpenXML(@pat, '/HL7/PATIENT', 1)
WITH (IDNum int GiNa varchar(20))
```

Figura 3-6. Exemplo de sintaxe do mapeamento OpenXML.

3.1.3. Oracle[4]

O SGBD Oracle, inclui mais de uma forma de suporte à XML: usando o tipo de dado XMLType; pelo utilitário XML SQL (XSU); e por uma visão canônica de XML sobre os dados relacionais.

O tipo de dados XMLType é usado para decompor documentos XML em um esquema objeto-relacional para armazenamento no banco. Pode ser usado para armazenar um documento inteiro ou partes do documento [MUR03]. O esquema objeto-relacional do banco pode ser definido a partir de um XML Schema.

O utilitário XML SQL (XSU) é uma interface para programação Java e PL/SQL para armazenamento e publicação de dados XML. XSU permite a aplicação de regras de mapeamento padronizadas ao resultado de uma consulta SQL. Não é possível definir um XML esquema para armazenamento, mas é possível gerar um DTD ou XML Schema para os dados publicados a partir do resultado da consulta.

---

[3] DOM - Document Object Model. http://www.w3.org/DOM/

[4] Oracle Support for XML. http://otn.oracle.com/tech/xml/xmldb/content.html.

A visão canônica de XML pode ser usada para consultar dados do banco. É uma visão virtual em XML do esquema completo do banco. O XML Schema desta visão é derivado do esquema objeto-relacional do próprio banco[AME03].

Na versão 9i Release 2, foi introduzido o Oracle XML DB. Mapeamentos são feitos de maneira automática, mas as regras podem ser sobrepostas por definições do usuário através de anotações em um XML Schema. O sistema inclui o XML Schema do usuário e guarda as informações de mapeamento internamente. A partir daí é possível gerar automaticamente tabelas, tipos SQL, índices e classes Java associadas ao mapeamento.

Dentre os SGBDs disponíveis no mercado, o Oracle é uma das mais atrativas opções para integração com XML devido às diversas opções que fornece para tratamento dos dados XML[AME03]. É importante destacar que o mapeamento que propõe não está baseado no modelo relacional, mas sim no modelo objeto-relacional incluindo o uso de características próprias do SGBD Oracle, o que torna difícil o aproveitamento do mapeamento gerado em outros SGBDs relacionais.

XML DB adiciona elementos e atributos novos no XML Schema para especificar o mapeamento. Entre os SGBDs comerciais é o único que se baseia no XML Schema [AME03].

```
Declare doc varchar(5000) :=
'<schema>
  <complexType name = "PATIENT" xdb:SQLType = "OBJ_T2">
    <sequence>
      <element name = "FaNa" type = "string"/>
      <element name = "OBX" xdb:SQLType = "CLOB">
    </sequence>
  </complexType>
</schema>'
Begin dbms_xmlschema.registerSchema(
'http://www.oracle.com/HL7.xsd', doc) end;
```

Figura 3-7. Exemplo de mapeamento Oracle. Fonte: [AME03]

A Figura 3-7 apresenta um exemplo de mapeamento para uso no SGBD Oracle. Após a compilação do esquema de mapeamento, os documentos podem ser inseridos de várias maneiras, como *drag-and-drop* entre janelas do windows, clientes FTP (*File Transfer Protocol)*, ou ainda os métodos convencionais do banco para inserção de informações. XML Schema é suportado diretamente pelo banco, portanto no momento

da inserção de documentos XML é possível validar o documento de acordo com o modelo XML Schema correspondente [AME03].

3.2. Conclusão

Em uma breve análise dos SGBDs estudados em relação a esta dissertação, pode-se perceber que existem desvantagens em usar estas linguagens proprietárias complexas para mapeamento: (i) elas são linguagens específicas para um único SGBD; (ii) o tempo adicional necessário para que os usuários aprendam a linguagem; (iii) como a linguagem permite qualquer tipo de mapeamento, torna-se difícil decidir que construção de mapeamento usar e; (iv) como transformações complexas são permitidas, o processo de decomposição dos documentos XML pode ter um custo elevado em termos de processamento e memória [AME04]. Por isto, a proposta deste trabalho usa regras de mapeamento que podem ser aplicadas sobre um esquema XML Schema, gerando um esquema relacional. Estas regras definem um mecanismo flexível de tradução e independente de plataforma.

4. Trabalhos Relacionados

De forma a alcançar o objetivo deste trabalho, deve-se investigar os trabalhos relacionados ao mapeamento XML→relacional com o intuito de aproveitar e aprimorar pesquisas já realizadas, definindo uma proposta de mapeamento mais completa possível entre os modelos. Este capítulo apresenta uma análise de alguns trabalhos existentes e ao final uma conclusão.

4.1. Trabalho de Varlamis

Em trabalhos existentes, como [VAR04], é comum abordar o tema de transformação entre modelos de dados diferentes utilizando-se do código XML-Schema e da linguagem SQL para apresentar as conversões propostas. No trabalho em questão, é apresentado o sistema X-Database, onde são apresentadas heurísticas para converter um documento XML-Schema, contendo metadados do formato de vídeo MPEG 7, para o modelo relacional, apresentando o resultado em SQL, para execução em um banco de dados relacional. Propõe também um mecanismo para atualizações e consultas através de documentos XML contendo os comandos a serem executados no banco de dados relacional. Todo o trabalho está baseado no uso do banco de dados Oracle 8i. A Figura 4-1 apresenta a arquitetura do sistema.

As regras de conversão transformam definições de XML-Schema em SQL, como pode ser visto no exemplo da Figura 4-2. Esta abordagem impede que se façam adaptações ou melhorias na conversão, pois apresenta uma saída final já na linguagem SQL.

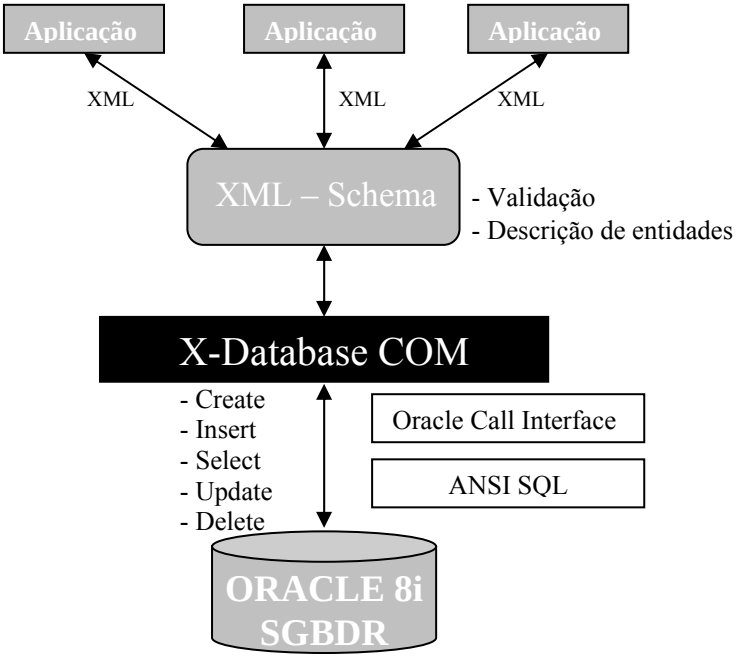


Figura 4-1. Arquitetura do sistema X-Database. Fonte [VAR04]

`<xs:complexType name="DigitalStorageDS">`
`  <xs:attribute name="id" type="ID"`
`  use="required"/>`
`  <xs:attribute name="name" type="string"`
`  use="required"/>`
`</xs:complexType>`
	CREATE TABLE DigitalStorageDS (
	id NUMBER NOT NULL,
	name VARCHAR2(20) NOT NULL,
	PRIMARY KEY (id));

Figura 4-2. Mapeamento de XML-Schema para SQL. Fonte: [VAR04]

As regras usadas pelo trabalho são:

a) Atributos e grupos de atributos – mapeados para um campo(coluna) ou conjunto de campos respectivamente;

b) Tipos simples – mapeado para um campo com restrições associadas ao conteúdo do mesmo;

c) Tipos complexos – cada tipo complexo é mapeado para uma tabela separada. Tabelas adicionais são criadas para representar relações muitos-para-muitos. Considera ainda neste item as seguintes situações para tipos complexos A,B,C,D e E: A possui exatamente uma referência para C; A possui referências ilimitadas para D; A contém exatamente um elemento do tipo B; A contém um número ilimitado de elementos do tipo E;

d) Tratamento de herança de tipos – cria uma tabela chamada OBJECTIDS para registrar id e type de cada elemento que participa na hierarquia de herança e define um grupo de *triggers* para garantir referências corretas entre os elementos desta hierarquia.

Nota-se que as regras propostas atendem o contexto usado como exemplo no trabalho, ou seja, metadados do formato de vídeo MPEG 7, deixando de apresentar os tratamentos necessários para situações onde são usados conceitos como *choice*, *group*, *susbtitutionGroup*, entre outros.

4.2. Trabalho de Kappel

[KAP04] também trata da integração de XML com bancos de dados relacionais, usando como entrada um esquema XML, especificado com o uso de DTD, gerando como saída o esquema relacional. O trabalho apresenta padrões para mapeamento entre XML e bancos de dados relacionais. Ele divide estes padrões de mapeamento em duas categorias: tipos básicos de mapeamentos e mapeamentos admissíveis(*reasonable*). Na categoria de mapeamentos básicos, são listados tipos de conversão "direta" e as possibilidades são: elemento convertido para uma relação(ET_R), elemento convertido para um atributo relacional(ET_A), e atributo convertido para um atributo relacional(A_A). A Figura 4-3 ilustra estes tipos de conversão.

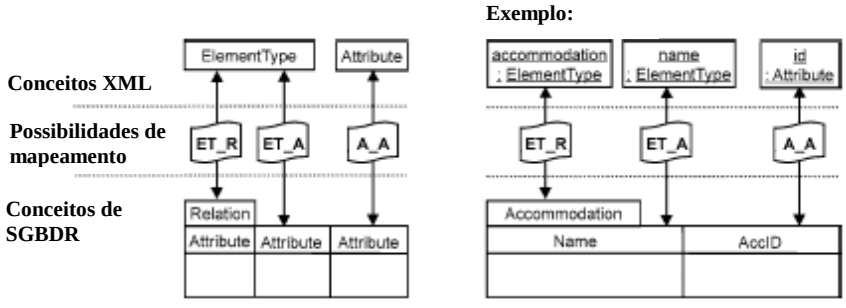


Figura 4-3. Tipos básicos de mapeamento. Fonte: [KAP04]

Já na categoria de mapeamentos admissíveis, são introduzidos os conceitos de mapeamentos diretos e indiretos para os tipos básicos de mapeamento gerando assim as

possibilidades: $ET\_R_{direto/indireto}$, $ET\_A_{direto/indireto}$, $A\_A_{direto/indireto.}$ Mapeamentos diretos são aqueles em que o valor mapeado é armazenado diretamente na relação que representa o elemento no qual este valor é contido. No mapeamento indireto o valor mapeado é armazenado em uma segunda relação e referenciado na relação que representa o elemento que o contém através de uma chave estrangeira.

4.3. Trabalho de Amer-Yahia

O trabalho apresentado por [AME04] propõe um *framework* para mapeamento de XML para relacional, onde este mapeamento é feito através de anotações no próprio modelo XML Schema que posteriormente é processado e como resultado são gerados comandos SQL para a criação das tabelas. O framework para definição de mapeamentos (Mapping Definition Framework - MDF) baseia-se em anotações realizadas no modelo XML Schema de entrada. Estas anotações são pré-definidas e em número limitado, controlando desta maneira os mapeamentos que podem ser definidos pelo usuário.

Um mapeamento é definido por uma quádrupla (XS, EM, AM, SM), onde:

- XS : XMLSchema de entrada;
- EM: Element mapping, é uma função que mapeia um elemento em uma tabela, coluna ou CLOB no esquema relacional;
- AM: Attribute mapping, é uma função que mapeia um atributo em uma tabela, coluna ou CLOB no esquema relacional;
- SM: Structure mapping, define uma identidade, estrutura e ordem de mapeamento.

As anotações no documento XML Schema de entrada, que definem o mapeamento, podem ser associadas a atributos, elementos e grupos. A sintaxe destas anotações consiste em adicionar atributos de um espaço de nomes chamado MDF em um documento XML Schema de entrada. Estes atributos são apresentados na Tabela 4-1.

Tabela 4-1. Atributos de anotação MDF.

Atributos para Anotação	Alvo	Valor	Ação

outline	atributo ou elemento	verdadeiro, falso	Se verdadeiro, uma tabela relacional é criada, caso contrário o mapeamento é feito para coluna(s) na tabela que o contém.
tablename	atributo, elemento ou grupo	String	O string é usado como nome para tabela
columname	atributo, elemento de tipo simples	String	O string é usado como nome para a coluna
sqltype	atributo, elemento de tipo simples	String	O string sobre-escreve o tipo SQL de uma coluna
structurescheme	elemento raiz	KFO, Interval, Dewey	Especifica o mapeamento da estrutura
edgemapping	Elemento	verdadeiro, falso	Se verdadeiro o elemento e seus descendentes são mapeados de acordo com a técnica Edge Mapping.
maptoclob	atributo ou elemento	verdadeiro, falso	Se verdadeiro o elemento ou atributo é mapeado para uma coluna tipo CLOB

Após a anotação destes atributos um processador de anotações realiza o *parsing* do documento XML Schema com anotações checando se o mapeamento é correto e sem perdas e gera um repositório de mapeamento e produzindo os comandos *CREATE TABLE* necessários para a construção do esquema relacional.

O sistema proposto ainda cobre a conversão de consultas sobre os dados armazenados como pode ser visto na Figura 4-4.

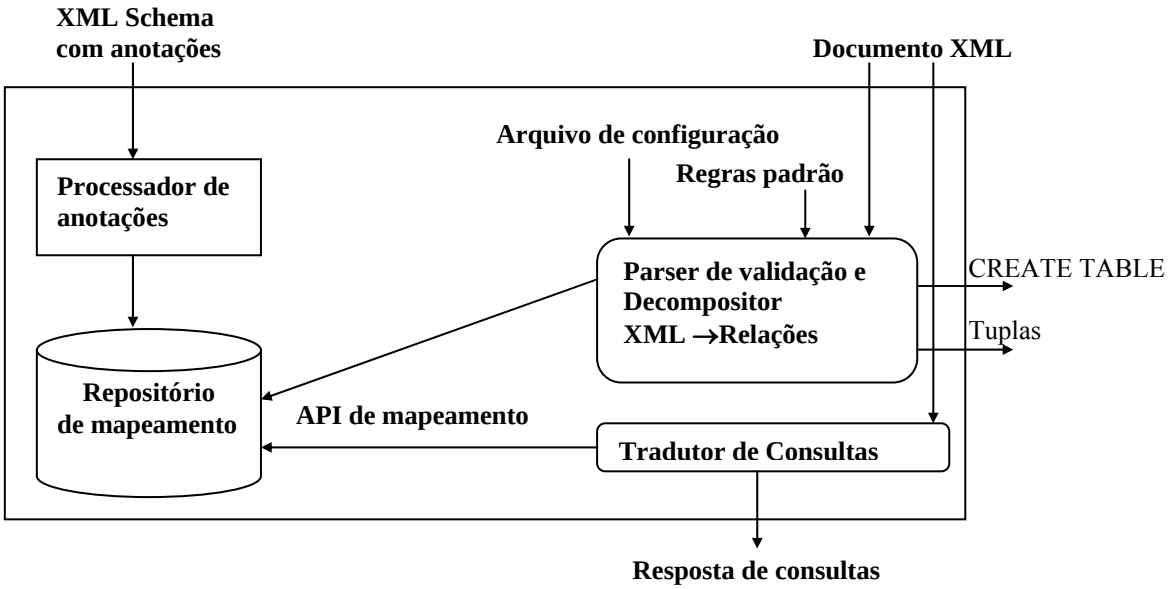


Figura 4-4. Arquitetura MDF. Fonte: [AME04]

4.4. Trabalhos com mapeamento no sentido Relacional → XML

Os trabalhos de [FON03], [DUT04] e [FER00] não têm uma correspondência direta com esta dissertação pois tratam do mapeamento do modelo relacional para o formato XML, ou seja no sentido contrário do que propõe este trabalho. Porém, devido à relevância das abordagens no contexto de mapeamento entre modelos de dados, eles foram analisados.

Em [FON03], o mapeamento relacional para XML é feito através dos seguintes passos: 1) união das tabelas num processo de *"denormalization"*; 2) integração das tabelas unificadas em uma árvore DOM; 3) seleção do elemento raiz do XML; 4) integração das árvores DOM geradas. Da mesma maneira, [DUT04] propõe a conversão de esquemas relacionais para XML Schema, apresentando como vantagem o uso do esquema relacional ao invés da base de dados armazenada no SGBD dado que o esquema provê informações adicionais úteis para o processo de conversão.

Em [FER00] é proposto um *framework* para publicação de dados de um banco relacional em XML. Nesta abordagem o sistema converte automaticamente o esquema relacional existente em uma visão canônica em XML representando as tabelas e seus atributos em XML. Usando esta visão canônica, ele define uma visão virtual em XML, aplicando um sub-conjunto da linguagem XQuery, que é o esquema XML.

Posteriormente, as informações são acessadas pelo uso de consultas em XQuery para obter documentos XML a partir do banco de dados relacional.

4.5. Conclusão

De maneira geral, as abordagens utilizadas não permitem que o desenvolvedor empregue sua experiência para otimizar o modelo resultante da transformação ao SGBD relacional utilizado em sua empresa, visto que o resultado do mapeamento no sentido XML→relacional, gera especificações em SQL na maioria das abordagens. Para evitar este tipo de problema, neste trabalho optou-se por realizar o mapeamento no nível lógico, abstraindo detalhes de representação física, ou seja, implementação do banco de dados relacional. Outro problema encontrado em alguns trabalhos é a consideração parcial dos conceitos do XML Schema, desconsiderando parte dos conceitos de maneira a simplificar o processo de mapeamento. A apresentação das regras de mapeamento se dá de maneira implícita, muitas vezes embutida no código do sistema proposto, deixando pouca informação ao usuário sobre como cada conceito do modelo XML Schema é mapeado para o modelo relacional.

A Tabela 4-2 apresenta os trabalhos relacionados e os respectivos modelos de origem e os modelos resultantes após o mapeamento.

Tabela 4-2. Trabalhos relacionados: entradas e saídas

Trabalho	Entrada	Saída
[VAR04]	XSD	SQL – tabelas no BD
[KAP04]	DTD	Esquema relacional
[AME04]	XSD	SQL – tabelas no BD
[FON03]	Banco de dados existente	DTD
[DUT04]	Esquema relacional	XSD
[FER00]	Banco de dados existente	XSD
Proposta deste trabalho	XSD	Esquema relacional

Com base nas deficiências encontradas, este trabalho propõe uma abordagem que tem como principal diferencial, em relação aos trabalhos analisados, a entrada em XML Schema com uma análise detalhada dos conceitos do modelo e a saída no modelo relacional em nível lógico.

5. ABORDAGEM DE MAPEAMENTO XML-RELACIONAL

5.1. Motivação

Um desafio que surge com o uso extensivo da XML é o gerenciamento eficiente de documentos XML. Em vários ambientes computacionais que utilizam XML para esta troca e consulta de informações, a persistência de dados XML em um banco de dados relacional apresenta-se como uma opção interessante, pois aproveita a maturidade e confiabilidade da tecnologia de banco de dados relacionais como meio de armazenamento. Para viabilizar este armazenamento é necessário encontrar uma maneira de associar os dados XML com as estruturas providas pelo modelo relacional, mantendo a semântica das informações. Esta associação é comumente identificada como mapeamento lógico.

Neste contexto, este trabalho apresenta mecanismos para transformar um esquema de dados XML, definido com o uso de XML Schema [W3C04], para um esquema de dados relacional. A escolha de XML Schema se dá pelo fato desta ser a recomendação atual da W3C para representação de esquemas XML. Neste trabalho é feita uma análise detalhada dos conceitos desta recomendação, que é, atualmente, a que apresenta mais recursos para a definição de esquemas XML. Já o modelo relacional foi escolhido por ser um modelo amplamente utilizado no nível comercial.

A transformação de uma definição em XML-Schema em uma definição relacional traz diversas vantagens, como:

- possibilidade de integração de dados através da definição de protocolos XML para transferência de dados entre bancos de dados relacionais;
- minimização de custos, reduzindo o impacto na transformação de dados entre sistemas, evitando a re-implementação ou a manutenção dos mesmos;
- continuidade da utilização de SGBDs relacionais pelas aplicações, aproveitando a performance e estabilidade dos mesmos;
- redução de custos, evitando a aquisição de um novo SGBD com suporte à XML.

Na seqüência são detalhados as regras e o algoritmo de mapeamento.

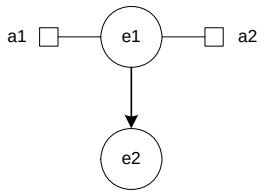
5.2. Princípios Gerais do Mapeamento

O objetivo deste trabalho, como afirmado anteriormente, é mapear uma definição em XML Schema para um esquema relacional, a nível lógico, apresentando de maneira detalhada as regras para mapeamento entre os modelos e o processo (algoritmo) de mapeamento para aplicação destas regras. Este mapeamento é exemplificado sobre as representações gráficas do modelo relacional, apresentada no capítulo 3, e do modelo XML Schema, apresentada em [MAU05] ANEXO 1. Desta forma, tem-se uma solução de mapeamento mais abstrata e flexível, permitindo ao desenvolvedor aplicar seus conhecimentos para obter um modelo físico mais otimizado para o seu SGBD relacional no momento da implementação. Outra vantagem está na clareza e abrangência das regras de mapeamento que exploram as vantagens de uma representação gráfica a nível lógico.

Para exemplificar o princípio geral de mapeamento para elementos e atributos, a Figura 5-1 apresenta o mapeamento de um elemento complexo contendo um elemento simples e dois atributos para o esquema relacional. Como pode ser visto na referida figura, de modo geral elementos complexos são convertidos em tabelas e elementos simples e atributos tornam-se colunas.

```
<xs:element name="e1">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="e2" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="a1" type="xs:ID"/>
        <xs:attribute name="a2" type="xs:integer"/>
    </xs:complexType>
</xs:element>
```

(a)



(b)

e1

a1	a2	e2

(c)

Figura 5-1. (a) definição XSD, (b) representação no modelo XML Schema, (c) esquema relacional

Alguns conceitos do modelo XML Schema possuem relação direta com o modelo relacional, como atributos de tipos atômicos e elementos simples com tipos simples. Entretanto, existem várias situações a serem consideradas no mapeamento

proposto, como ID no modelo XML e chaves no modelo relacional, atributos do tipo lista, elementos complexos de vários níveis, etc, que não possuem uma relação direta, exigindo uma análise aprofundada destes conceitos para se propor uma solução. Este trabalho leva em conta estas situações. Estas diferenças se devem ao fato de que XML Schema foi desenvolvida para representar dados com estruturas altamente complexas, incluindo várias formas de representação para estes dados, como por exemplo, dados do tipo lista. Porém, no modelo relacional as tabelas representam dados com estrutura mais simples, estando a sua maior vantagem na sua definição formal e na estabilidade e desempenho dos sistemas existentes, enquanto para XML ainda não existem sistemas maduros o suficiente para superar a estabilidade e desempenho dos SGBDs relacionais. Por isso, a abordagem proposta busca explorar os pontos fortes do XML e dos bancos de dados relacionais, oferecendo desta maneira uma opção interessante para o gerenciamento de dados XML descritos em XML Schema.

A ênfase do mapeamento proposto por este trabalho está na capacidade de mapear situações estruturais dos modelos XML Schema, mantendo esta estrutura no modelo relacional resultante. Por isto, para a definição das regras de mapeamento foi realizada uma análise aprofundada da linguagem XML Schema identificando os conceitos que afetam diretamente a estrutura do documento XML modelado. Estes conceitos são apresentados na Tabela 5-1.

Tabela 5-1. Conceitos XML Schema considerados

<table>
<tr><th colspan="4">Conceito XML Schema</th></tr>
<tr><td rowspan="4">Atributo</td><td colspan="3">Atômico</td></tr>
<tr><td colspan="3">Lista</td></tr>
<tr><td colspan="3">União</td></tr>
<tr><td colspan="3">Derivado</td></tr>
<tr><td rowspan="8">Elemento</td><td rowspan="3">Simples</td><td colspan="2">Atômico</td></tr>
<tr><td colspan="2">derivado por restrição</td></tr>
<tr><td colspan="2">Union</td></tr>
<tr><td rowspan="5">Complexo</td><td rowspan="4">conteúdo simples com atributos</td><td>tipo atômico</td></tr>
<tr><td>List</td></tr>
<tr><td>Union</td></tr>
<tr><td>Derivado</td></tr>
<tr><td rowspan="2">Vazio</td><td>Com atributos</td></tr>
<tr><td>Sem atributos</td></tr>
<tr><td rowspan="3">contendo outros elementos</td><td>Sequence</td></tr>
<tr><td>All</td></tr>
<tr><td>Choice</td></tr>
</table>

		Misto
	anyType	
Grupo		
Grupo de Atributos		
Grupo de Substituição		
Abstrato		

As próximas seções detalham as etapas que compreendem a abordagem de mapeamento proposta. Estas etapas são:

1. Pré-processamento: modificações iniciais visando adequar o modelo de entrada para realizar o mapeamento;
2. Regras de mapeamento e algoritmo:
    a. definição das regras de mapeamento que são aplicadas sobre as estruturas XML do esquema de entrada para gerar o esquema relacional;
    b. seqüência de passos ordenados para a aplicação das regras de mapeamento.

5.3. Pré-Processamento

Esta primeira etapa de pré-processamento tem por objetivo adequar o esquema XML de entrada para a etapa posterior, visando reduzir o número de regras de mapeamento a serem criadas. Estas adequações ocorrem através de algumas alterações na estrutura do esquema XML.

As alterações realizadas nesta etapa visam: (i) incorporar o conteúdo de documentos externos usados na definição do esquema, com recursos do grupo de gerenciamento da linguagem XML Schema; (ii) converter grupos de substituição em *choices*; (iii) substituir definições do tipo *extension* indicadas em um elemento pelo conteúdo a ser estendido e; (iv) eliminar referências. Desta forma, a definição de todas as estruturas do esquema se tornam explícitas nas definições dos elementos.

5.3.1. Incorporação de conteúdo externo

Uma definição de esquema pode ser especificada integralmente em um único arquivo, mas também pode ter base em definições de esquema que se encontrem em outros arquivos, inclusive com espaços de nomes diferentes.

Para construir esquemas com definições de arquivos externos pode-se usar os elementos *include*, *import* e *redefine*. O elemento *include* permite incluir definições externas, que estão no mesmo espaço de nome ou sem definição de espaço de nome, diretamente no esquema em construção; o seu efeito é similar a uma cópia de conteúdo. Quando o espaço de nome do esquema XML externo a ser usado é diferente deve-se usar o elemento *import*, que faz a inclusão de conteúdo levando em consideração os espaços de nome do modelo importado e do modelo que está importando. Por fim, o elemento redefine permite importar conteúdos externos e redefinir partes do modelo importado, ajustando o modelo de conteúdo ao modelo em construção.

Para realizar o mapeamento para o modelo relacional deve-se considerar o modelo com os conteúdos incluídos, importados ou redefinidos como pertencentes ao mesmo. Portanto, no modelo de entrada para o mapeamento deve-se considerar o modelo final, cuja estrutura já sofreu os efeitos da aplicação destes três elementos.

5.3.2. Tratamento de Grupos de Substituição

Para tratar grupos de substituição, compostos por um elemento "cabeça" e elementos que podem substituir este elemento "cabeça", deve-se realizar o seguinte: (i) criar um *choice* contendo o elemento "cabeça", nos locais onde o mesmo for usado no esquema; (ii) incluir uma cópia de cada elemento que pode substituir o "cabeça", dentro do *choice* criado para o mesmo; (iii) remover o atributo *substitutionGroup* da declaração dos elementos que podem substituir o "cabeça". Após estas alterações, o grupo de substituição será mapeado pela regra de mapeamento correspondente.

A Figura 5-2 apresenta a declaração de um grupo de substituição onde um elemento *pessoa2* pode ser usado em substituição ao elemento *pessoa1*. Ambos são derivações válidas de *pessoa*.

```
<xs:complexType name="pessoa">
  <xs:sequence>
    <xs:element name="Nome" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="sexo" type="xs:string"/>
</xs:complexType>

<xs:element name="pessoa1" type="pessoa"/>

<xs:element name="pessoa2" substitutionGroup="pessoa1">
<xs:complexType>
  <xs:complexContent>
    <xs:extension base="pessoa">
      <xs:sequence>
        <xs:element name="sobrenome" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
</xs:element>

<xs:element name="aluno">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="pessoa1"/>
      <xs:element name="registro" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figura 5-2. Declaração de grupo de substituição.

A Figura 5-3 apresenta o esquema resultante após a realização das alterações necessárias.

```
<xs:complexType name="pessoa">
  <xs:sequence>
    <xs:element name="Nome" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="sexo" type="xs:string"/>
</xs:complexType>

<xs:element name="pessoa1" type="pessoa"/>

<xs:element name="pessoa2">
<xs:complexType>
  <xs:complexContent>
    <xs:extension base="pessoa">
      <xs:sequence>
        <xs:element name="sobrenome" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
</xs:element>
```

```
<xs:element name="aluno">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element ref="pessoa1"/>
        <xs:element ref="pessoa2"/>
      </xs:choice>
      <xs:element name="registro" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figura 5-3. Criação de choices para tratar grupos de substituição.

A estrutura das instâncias resultantes do esquema da Figura 5-2 ou do esquema da Figura 5-3 são idênticas.

5.3.3. Tratamento de *Extensões*

A linguagem XML Schema permite a definição de extensões de conteúdos complexos, incluindo, por exemplo, tipos e elementos complexos, concretos ou abstratos. Os conteúdos abstratos são declarados com o uso do atributo *abstract* com valor *true*, fazendo com que o elemento não possa ser instanciado, podendo apenas ser estendido a partir da definição de outros tipos ou elementos. O conceito de extensão tem uma semântica semelhante ao conceito de herança, onde o elemento que faz a extensão passa a possuir o conteúdo do elemento do qual estende a definição, como pode ser visto na Figura 5-4.

```
<xs:complexType name="pessoa" abstract="true">
  <xs:sequence>
    <xs:element name="Nome" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="sexo" type="xs:string"/>
</xs:complexType>

<xs:element name="pessoa2">
<xs:complexType>
  <xs:complexContent>
    <xs:extension base="pessoa">
      <xs:sequence>
       <xs:element name="sobrenome" type="xs:string"/>
```

```
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
</xs:element>
```

Figura 5-4. Declaração de extensão.

Para realizar esta transformação, deve-se substituir as referências à estrutura estendida (atributo `base`) pelo conteúdo herdado, gerando um esquema XML no qual os elementos incorporam este conteúdo. A Figura 5-5 apresenta o resultado da substituição da extensão de tipos abstratos, mostrada na Figura 5-4.

```
<xs:element name="pessoa2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Nome" type="xs:string"/>
      <xs:element name="sobrenome" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="sexo" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Figura 5-5. Exemplo de incorporação de conteúdo estendido.

5.3.4. Eliminação de Referências

Uma definição de esquema de dados em XML Schema permite a definição de estruturas globais, que são declaradas como filhas do elemento *schema*. Estas declarações definidas pelo usuário podem ser referenciadas a partir da definição de outras estruturas com o uso dos atributos *ref* ou *type*, como pode ser visto na Figura 5-6.

O objetivo de se fazer referências a declarações globais, que podem envolver elementos, grupos de elementos, atributos, grupos de atributos e tipos, é reaproveitar a definição das estruturas já declaradas globalmente. Com isso, obtém-se um esquema mais conciso e que evita possíveis erros na redefinição de estruturas já declaradas.

```
<schema>
  <xsd:element name="comment" type="xsd:string"/>
  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      ...
      <xsd:element ref="comment" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name=”purchaseOrder” type=” PurchaseOrderType”/>
</schema>
```

Figura 5-6. Referência à declarações globais

Para realizar o mapeamento destes esquemas é necessário considerar a estrutura final dos elementos que serão mapeados. Portanto, deve-se substituir as referências encontradas pelo conteúdo referenciado, eliminando assim o reaproveitamento de código.

Após a realização das substituições, as declarações de grupos de elementos, grupos de atributos e tipos globais nomeados devem ser removidas, visto que seu conteúdo já foi incorporado nas definições de elementos. A Figura 5-7 apresenta o resultado da substituição da referência mostrada na Figura 5-6.

```
<schema>
  <xsd:element name="comment" type="xsd:string"/>

  <xsd:element name=”purchaseOrder”>
    <xsd:sequence>
      ...
      <xsd:element name="comment" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:element>
</schema>
```

Figura 5-7. Eliminação de referências.

5.4. Regras de Mapeamento

Em se tratando de mapear as estruturas existentes em um esquema XML Schema, é possível identificar várias possibilidades de correspondência no modelo relacional para um mesmo conceito do modelo XML Schema. Para garantir maior flexibilidade e clareza, as possibilidades mais significativas foram identificadas e apresentadas em cada regra como alternativas de mapeamento, ficando o desenvolvedor livre para escolher a opção que melhor se ajuste ao seu problema.

Para melhor entender a definição das regras de mapeamento a notação usada nas mesmas é descrita a seguir.

• opção 1
  • sub-item 1
  • sub-item 2
  • ...
  • sub-item n
• opção 2
• ...
• opção n
	- As chaves contêm um conjunto de opções possíveis das quais apenas uma deve ser selecionada. A leitura do conteúdo da mesma deve ser: opção 1 OU opção 2 OU opção n.
	- As sub-listas de itens são componentes de uma opção e por isso quando esta opção é escolhida todos os seus componentes devem ser executados. Sua leitura deve ser: sub-item 1 E sub-item 2 E sub-item n.

A seguir são apresentadas as definições das regras de mapeamento dos conceitos listados na Tabela 5-1 juntamente com exemplos. Estas regras são invocadas por um algoritmo de mapeamento, que constitui esta segunda etapa do processo.

5.4.1. Atributo de Tipo Atômico

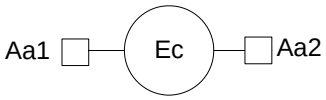
Esta regra considera que um atributo de tipo atômico pode ser mapeado para uma coluna na tabela correspondente ao elemento que o contém.

Definição: Um atributo atômico *AT* de nome *Aa* e tipo de dado *Ta* contido em um elemento complexo *EC* gera uma coluna de nome *Aa* e tipo de dado *Ta* na tabela que mantém os dados de *EC*. Se *AT* for do tipo *ID* ou *IDREF*, a coluna gerada deve ser indicada como chave primária e chave estrangeira respectivamente.

Exemplo:

```
<xs:element name="Ec">
 <xs:complexType>
  ...
  <xs:attribute name="Aa1" type="Ta"/>
  <xs:attribute name="Aa2" type="Ta"/>
 </xs:complexType>
</xs:element>
```

(a)



(b)

Ec

cp	Aa1	Aa2

(c)

Figura 5-8. Exemplo de Mapeamento de Atributo Atômico: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

5.4.2. Atributo de Tipo *List*

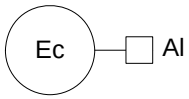
Esta regra considera que um atributo de tipo *list* pode ser mapeado para uma tabela própria, que armazena o conteúdo da lista e faz referência à tabela correspondente ao elemento que contém o atributo.

Definição: Um atributo do tipo *list ATL* de nome *Al* e tipo de dado da lista *Tl* contido em um elemento complexo *E* de nome *Ec*, gera uma tabela de nome *tab_Ec_Al*, contendo duas colunas: uma sendo a chave estrangeira *ce*, que referencia a tabela que mantém os dados de *E*, e a outra sendo uma coluna de nome *Al* e tipo de dado *Tl* que contém um item de *Al* [5]. Se *ATL* for do tipo *IDREFS*, a coluna *Al* gerada deve ser indicada como chave estrangeira.

Exemplo:

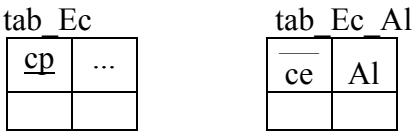
```
<xs:element name="Ec">
 <xs:complexType>
  ...
  <xs:attribute name="Al">
    <xs:simpleType>
      < xs:list itemType=”Tl”/>
    </ xs:simpleType>
  </xs:attribute>
</xs:complexType>
</xs:element>
```

(a)



(b)

[5] Um ATL com n itens sempre gera n linhas em tab_EC_Al, uma linha para cada item de Al.



(c)

Figura 5-9. Exemplo de Mapeamento de Atributo do tipo List: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

5.4.3. Atributo de Tipo *Union*

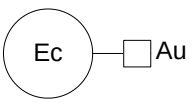
Esta regra considera que um atributo do tipo *union* pode ser mapeado para uma coluna com o tipo mais abrangente da *union* ou uma coluna do tipo texto, na tabela correspondente ao elemento que o contém, ou ainda, para uma tabela própria com uma coluna de referência para a tabela correspondente ao elemento que contém o atributo, com colunas para os tipos de dados pertencentes à *union*.

Definição: Um atributo do tipo *union TU* de nome *Au* contido em um elemento complexo *Ec*, com um conjunto de tipos de dados da *union* $tu_1,tu_2,...,tu_n$, gera:

- uma coluna de nome A*u* que armazena o conteúdo de *Tu* na tabela mantém os dados de *Ec*, e cujo tipo *T* seja o mais genérico dentre $tu_1,tu_2,...,tu_n$, caso *T* englobe todos os valores admissíveis por *Tu*;
- uma coluna de nome *Au* na tabela que mantém os dados de *Ec*, e tipo de dado texto, que armazena o conteúdo de *Tu*;
- uma tabela de nome *tab_Ec_TU* com colunas de nome $C_1, C_2,...,C_n$ com tipos correspondentes a cada item do conjunto $tu_1,tu_2,...,tu_n$ e uma coluna chave estrangeira que referencia a tabela que mantém os dados de *Ec*. Definir restrições de integridade para garantir que em cada linha da tabela somente uma coluna em $C_1, C_2,...,C_n$ possua valor.

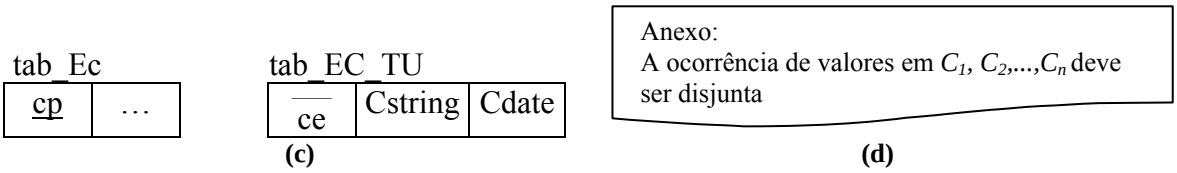
Exemplo:

```
<xs:element name="Ec">
 <xs:complexType>
  ...
  <xs:attribute name="Au">
    <xs:simpleType>
      <xs:union memberTypes=″xs:string xs:date″/>
    </xs:simpleType>
  </xs:attribute>
```



```
 </xs:complexType>
</xs:element>
```

(a) **(b)**



(c) **(d)**

Figura 5-10. Exemplo de Mapeamento de Atributo tipo Union: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional; (d) anexo ao esquema relacional descrevendo restrições de integridade.

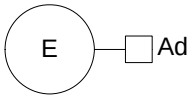
5.4.4. Atributo de Tipo Derivado

Esta regra considera que um atributo de tipo derivado é mapeado para uma coluna na tabela correspondente ao elemento que o contém.

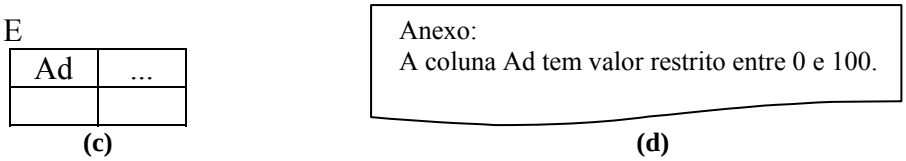
Definição: Um atributo *A* de tipo de dado derivado *Td* de nome *Ad*, contido em um elemento *E*, gera uma coluna de nome *Ad* e tipo de dado *Td* na tabela que mantém os dados de *E*, e um anexo com uma restrição de integridade correspondente à derivação de *Td*.

Exemplo:

```
<xs:element name="E">
 <xs:complexType>
  ...
  <xs:attribute name="Ad">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="100"/>
      </xs:restriction>
    </xs:simpleType>
 </xs:complexType>
</xs:element>
```



(a) **(b)**



(c) **(d)**

Figura 5-11. Exemplo de Mapeamento de Atributo derivado: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional; (d) anexo ao esquema relacional descrevendo restrições de integridade.

5.4.5. Elemento Simples de Tipo Nativo

Esta regra considera que um elemento simples do tipo nativo pode ser mapeado para uma tabela própria ou para uma coluna na tabela correspondente ao seu elemento ancestral. A regra considera ainda múltiplas ocorrências do elemento no seu elemento ancestral, definindo, neste caso, sempre uma tabela para ele.

A existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento simples, é também considerada. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido[6].

Definição: Um elemento simples *ES* de tipo nativo de nome *Esn* e tipo de dado *Tesn*, com cardinalidade máxima *cm*:

-
  -
    - gera uma coluna de nome *Esn* e tipo de dado *Tesn* em *tab_Anc*, se *ES* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc;*
    - gera uma tabela de nome *tab_Anc_Esn* com uma coluna de nome *Esn* e tipo de dado *Tesn* que armazena o conteúdo do elemento, se *ES* não possui um elemento ancestral;

  , se *ES* possui *cm* <= 1;

  - gera uma tabela de nome *tab_Anc_Esn* com uma coluna de nome *Esn* e tipo de dado *Tesn* que armazena o conteúdo do elemento. Inclui-se uma coluna chave estrangeira *ce* em *tab_Anc_Esn* que referencia *tab_Anc,* se *ES* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*

  , se *ES* possui *cm* > 1.

- gera:
  - uma coluna de nome *ordem_Esn* na tabela *tab* que mantém o conteúdo de *ES,*

- a chave primária de *tab* é o par *(ordem_Esn, ce)*, se *tab* ≠ *tab_Anc*.

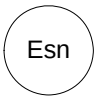
, se *ES* está contido em um indicador de ordem *sequence* em um elemento ancestral.

[6] Abordagem usada em [AME04] para tratamento de indicadores de ordem *sequence* em esquemas relacionais.

Exemplo:

```
<xs:element name="Esn"
 type=″xs:string″ maxOccurs=″5″>
  ...
</xs:element>
```

(a)



(b)

tab_Anc

cp	...

tab_Anc_Esn

ce	ESN

(c)

Figura 5-12. Exemplo de Mapeamento de Elemento Simples: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

5.4.6. Elemento Simples Derivado por Restrição

Esta regra considera que um elemento simples de tipo derivado por restrição pode ser mapeado para uma tabela própria ou para uma coluna na tabela correspondente ao seu elemento ancestral. A regra considera ainda múltiplas ocorrências do elemento no seu elemento ancestral, definindo, neste caso, sempre uma tabela para ele.

A existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento simples, é também considerada. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido.

Definição: Um elemento simples *ES* de tipo de dado derivado por restrição *Tdr* de nome *Edr*, com cardinalidade máxima *cm*:

- se *ES* possui *cm* <= 1;
  - gera uma tabela de nome *tab_Anc_Edr* com uma coluna de nome *Edr* de tipo *Tdr* que armazena o conteúdo de *ES*. Incluir uma coluna chave estrangeira *ce* em *tab_Anc_Edr* que referencia *tab_Anc, s*e *ES* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc* ;
  - gera uma coluna de nome *Edr* e tipo de dado *Tdr* em *tab_Anc,* se *ES* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*;
- se *ES* possui *cm* > 1.
  - gera uma tabela de nome *tab_Anc_Edr* com uma coluna de nome *Edr* que armazena o conteúdo do elemento. Incluir uma coluna chave estrangeira *ce* em *tab_Edr* que referencia *tab_Anc, s*e *ES* possui um elemento ancestral que tenha sido mapeado para uma tabela *tab_Anc*

- gera um anexo com uma restrição de integridade correspondente à *Tdr.*
- gera:
  - uma coluna de nome *ordem_Edr* na tabela *tab* que mantém o conteúdo de *ES*;
  - a chave primária de *tab* é o par *(ordem_Edr, ce)*, se *tab* ≠ *tab_Anc*.

, se *ES* está contido em um indicador de ordem *sequence* em um elemento ancestral.

Exemplo:

```
<xs:element name="Edr">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxExclusive value="999"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

(a)

(b)

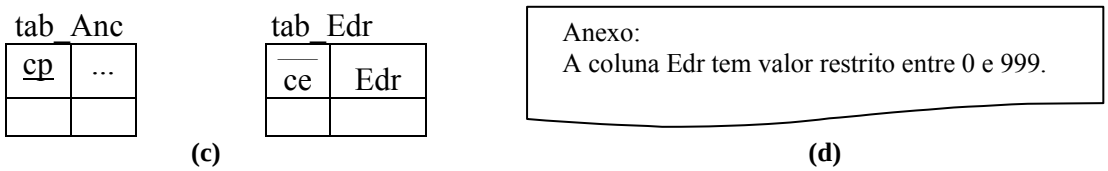


Figura 5-13. Exemplo de Mapeamento de Atributo derivado: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional; (d) anexo ao esquema relacional descrevendo restrições de integridade.

5.4.7. Elemento Simples de Tipo *Union*

Esta regra considera que um elemento simples do tipo *union* pode ser mapeado para uma tabela com: (i) uma coluna com o tipo mais abrangente da *union*; (ii) uma coluna do tipo texto ou, (iii) colunas para os tipos de dados pertencentes à *union*.

Em todos os casos de mapeamento, deve-se incluir uma coluna de referência para a tabela correspondente ao elemento ancestral, se este existir.

A existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento simples, é também considerada. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido.

Definição: Um elemento simples *ES* de nome *Esu* e de tipo *union TU,* com um conjunto de tipos de dados da *union* $tu_1,tu_2,...,tu_n$:

- gera uma tabela de nome *tab_Anc_Esu* com uma coluna de nome *Esu* que armazena o conteúdo de *ES* e cujo tipo *T* seja o tipo mais genérico dentre $tu_1,tu_2,...,tu_n$, caso *T* englobe todos os valores admissíveis por *TU*;
- gera uma tabela de nome *tab_Anc_Esu* com uma coluna de nome *ESU* e tipo de dado texto que armazena o conteúdo de *ES*;
- gera uma tabela de nome *tab_Anc_Esu* com colunas de nome $c_1, c_2,...,c_n$ com tipos correspondentes a cada item do conjunto $tu_1,tu_2,...,tu_n$. Define-se restrições de integridade para garantir que em cada linha da tabela somente uma coluna em $c_1, c_2,...,c_n$ possua valor.

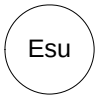
- gera uma coluna chave estrangeira *ce* em *tab_Anc_Esu* que referencia *tab_Anc*, se *ES* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc.*
- gera:
  - uma coluna de nome *ordem_Esu* em *tab_Anc_Esu*;
  - a chave primária de *tab_Anc_Esu* é o par *(ordem_Esu, ce)*

, se *ES* está contido em um indicador de ordem *sequence* em um elemento ancestral.

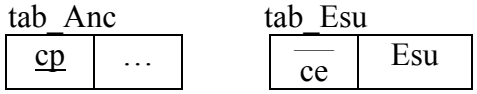
Exemplo:

```
<xs:element name="Esu">
  <xs:simpleType>
    <xs:union memberTypes="  xs:integer
                             xs:date
                             xs:string"/>
  </xs:simpleType>
</xs:element>
```

(a)



(b)



(c)

Figura 5-14. Exemplo de Mapeamento de Elemento tipo Union: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

5.4.8. Elemento Complexo

Esta regra considera que um elemento complexo é mapeado para uma tabela própria. A regra considera ainda o uso do indicador de ordem *choice* no elemento complexo, definindo, neste caso, restrições de integridade para o mapeamento.

A existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento complexo, é também considerada. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido.

Definição: Um elemento complexo *EC* de nome *Ecs* e tipo *TEcs*:

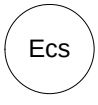
- gera uma tabela de nome *tab_Anc_Ecs* com uma coluna chave primária C*p*;
- gera uma coluna chave estrangeira *ce* em *tab_Anc_Ecs* que referencia *tab_Anc*, se *EC* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*;
- gera restrições de integridade referentes a uma derivação, se *TEcs* é um tipo derivado por restrição;
-
  - gera uma coluna que armazena o conteúdo de *Ecs*, se *EC* possui conteúdo simples;
  - gera restrições de integridade para garantir que, a cada ocorrência de *Ecs*, somente um elemento entre $e_1, e_2, ..., e_n$, possua valor, se *EC* usa indicador de ordem *choice* e contém elementos $e_1, e_2, ..., e_n$.
- gera:
  - uma coluna de nome *ordem_Ecs* em *tab_Anc_Ecs*;
  - a chave primária de *tab_Anc_Ecs* é o par *(ordem_Ecs, ce)*

, se *EC* está contido em um indicador de ordem *sequence* em um elemento ancestral.

Exemplo 1 – Elemento complexo com conteúdo simples:

```
<element name= “Ecs”>
 <complexType>
  <simpleContent>
   <extension base=“string”>
     ...
   </extension>
  </simpleContent>
 </complexType>
</element>
```

(a)



(b)

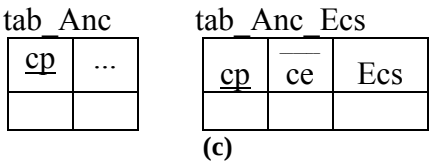
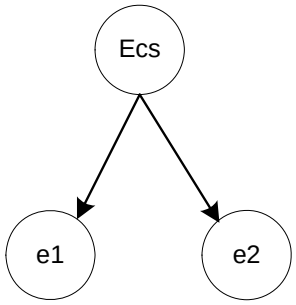


Figura 5-15. Exemplo de Mapeamento de Elemento complexo: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

Exemplo 2 – Elemento complexo com indicador *sequence*

```
<xs:element name="Ecs">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="e1" type="xs:integer"/>
      <xs:element name="e2" type="xs:string"/>
      ...
    </xs: sequence>
  </xs:complexType>
</xs:element>
```



(a) **(b)**

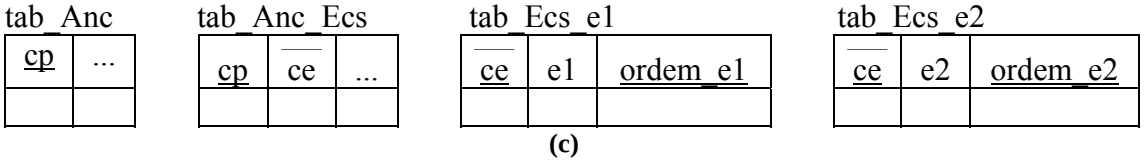
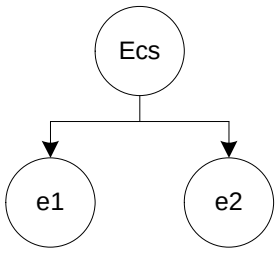


Figura 5-16. Exemplo de Mapeamento de Elemento complexo com indicador *sequence*: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional; (d) anexo ao esquema relacional descrevendo restrições de integridade.

Exemplo 3 – Elemento complexo com indicador *choice*

```
<xs:element name="Ecs">
  <xs:complexType>
    <xs:choice>
      <xs:element name="e1" type="xs:integer"/>
      <xs:element name="e2" type="xs:string"/>
      ...
    </xs:choice>
  </xs:complexType>
</xs:element>
```



(a) **(b)**

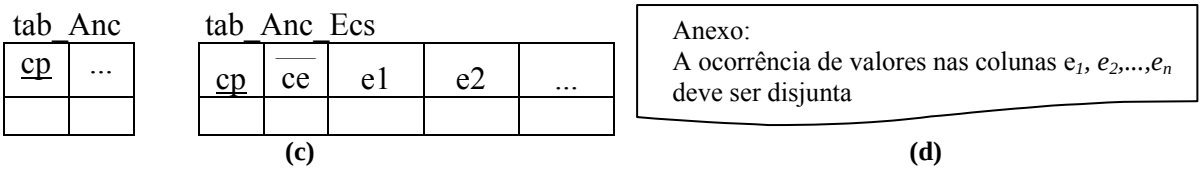


Figura 5-17. Exemplo de Mapeamento de Elemento complexo com indicador *choice*: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional; (d) anexo ao esquema relacional descrevendo restrições de integridade.

5.4.9. Elemento Complexo Vazio

Esta regra considera que um elemento complexo vazio com atributos pode ser mapeado para uma tabela própria, e caso isto aconteça, a existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento complexo, é também considerada. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido.

A regra considera ainda que um elemento complexo vazio sem atributos pode não ter nenhum mapeamento ou então ser mapeado para uma coluna na tabela correspondente ao seu elemento ancestral. Neste segundo caso, o número de ocorrências do elemento é armazenado nesta coluna.

Definição: Um elemento complexo *EC* vazio de nome *Ecv*:

- gera:
  - uma tabela *tab_Anc_Ecv*;
  - uma coluna chave estrangeira *ce* em *tab_Anc_Ecv* que referencia *tab_Anc,* se *ES* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*
  - e:
    - uma coluna de nome *ordem_Ecv* em *tab_Anc_Ecv*;
    - a chave primária de *tab_Anc_Ecv* é o par *(ordem_Ecv, ce)*

  , se *EC* está contido em um indicador de ordem *sequence* em um elemento ancestral

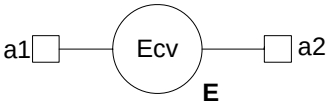
, se *Ecv* possui atributos.

-
  - nenhum mapeamento;
  - uma coluna numérica em *tab_Anc*, que mantém o número de ocorrências de *Ecv,* se *Ecv* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*.

  , se *Ecv* não possui atributos.

Exemplo 1 – Elemento complexo vazio com atributos:

```
<xs:element name="Ecv">
  <xs:complexType>
    <xs:attribute name="a1" type="xs:string"/>
    <xs:attribute name="a2" type="xs:string"/>
    ...
  </xs:complexType>
</xs:element>
```



(a) **(b)**

tab_Anc

cp	...

tab_Anc_Ecv

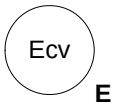
ce	a1	a2	...

(c)

Figura 5-18. Exemplo de Mapeamento de Elemento complexo vazio com atributos: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

Exemplo 2 – Elemento complexo vazio sem atributos:

```
<xs:element name="Ecv">
  <xs:complexType></xs:complexType>
</xs:element>
```



(a) **(b)**

tab_Anc

cp	...	Ecv

(c)

Figura 5-19. Exemplo de Mapeamento de Elemento complexo vazio sem atributos: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

5.4.10. Elemento Complexo de Tipo Misto

Esta regra considera que um elemento complexo de tipo misto pode ser mapeado para uma tabela própria, com ou sem uma coluna de texto longo para armazenar todo o conteúdo do elemento misto (textos e estruturas), e uma tabela que comporta somente a parte textual do elemento. Outra alternativa é o mapeamento do elemento para uma única tabela com uma coluna para armazenar o todo o conteúdo do elemento.

A existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento complexo, é também considerada. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido.

Considera-se ainda que outras regras são posteriormente aplicadas para o mapeamento dos elementos componentes do elemento misto. Tal tratamento, apesar de gerar redundância no armazenamento dos dados XML, permite que se possa recuperar a estrutura do elemento misto na íntegra (mantida na coluna de texto longo) e, ao mesmo tempo, manter diretamente no esquema relacional as relações hierárquicas do elemento misto com os seus elementos componentes, para facilitar consultas.

Definição: um elemento complexo *EC* de tipo misto *Tm* de nome *Ecm* contendo elementos e_1, e_2, ..., e_n:

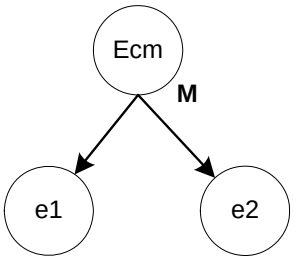
- gera:
  - gera:
    - uma tabela de nome *tab_Anc_Ecm* com chave primária *cp* e uma coluna *Ecm* tipo texto longo;
    - uma tabela de nome *tab_Anc_Ecm* com chave primária *cp*;
  - uma tabela de nome *tab_Anc_Ecm_Tp* com uma coluna *tp* de tipo texto para armazenar o conteúdo texto puro de *Ecm* e uma coluna chave estrangeira *ce* que referencia *tab_Anc_Ecm*;
- gera uma tabela de nome *tab_Anc_Ecm* com chave primária *cp* e uma coluna *Ecm* tipo texto longo;

- gera uma coluna chave estrangeira *ce* em *tab_Anc_Ecm* que referencia *tab_Anc,* se *Ecm* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*.
- gera:
  - uma coluna de nome *ordem_Ecm* em *tab_Anc_Ecm*;
  - a chave primária de *tab_Anc_Ecm* é o par *(ordem_Ecm, ce)*

, se *EC* está contido em um indicador de ordem *sequence* em um elemento ancestral.

Exemplo:

```
<xs:element name="Ecm">
 <xs:complexType mixed="true">
  <xs:sequence>
   <xs:element name="e1" type="xs:string"/>
   <xs:element name="e2" type="xs:integer"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```



(a) **(b)**

tab_Anc

cp	...

tab_Anc_Ecm

ce	tl	e1	ordem_e1	e2	ordem_e2

tab_Anc_Ecm_Tp

ce	tp

(c)

Figura 5-20. Exemplo de Mapeamento de Elemento complexo de tipo misto: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

5.4.11. Elemento de Tipo AnyType

Esta regra considera que um elemento de tipo *anyType* pode ser mapeado para uma tabela própria com uma coluna para armazenar todo o conteúdo do elemento na forma original, ou uma coluna na tabela correspondente ao seu elemento ancestral, e mais uma tabela para os possíveis atributos do elemento.

A regra considera ainda, a existência de um indicador de ordem *sequence*, na definição do elemento ancestral que contém o elemento *anyType*. Neste caso, um atributo adicional que indica a ordem de ocorrência do elemento é definido.

Definição: um elemento *E* de tipo *anyType Ta* de nome *Ea* contendo elementos e_1, e_2, ..., e_n e atributos de nomes a_1, a_2, ..., a_n com conteúdos ca_1, ca_2, ..., ca_n respectivamente:

- gera uma tabela de nome *tab_Anc_Ea* com chave primária *cp* e uma coluna de tipo texto longo;
- gera uma coluna de nome *Ea* de tipo texto longo em *tab_Anc*, se *E* possui um elemento ancestral *Anc* que tenha sido mapeado para uma tabela *tab_Anc*;

- gera uma tabela de nome *tab_Atributos_Ea* com uma coluna de nome *nomeAtributo* para armazenar os nomes a_1, a_2, ..., a_n, outra coluna de nome *conteúdoAtributo* para armazenar ca_1, ca_2, ..., ca_n, e uma coluna chave estrangeira que referencia tabela que mantém *Ea*.[7]
- gera:
  - uma coluna de nome *ordem_Ea* na tabela *tab* que mantém o conteúdo de *E*;
  - a chave primária de *tab* é o par *(ordem_Ea, ce)*, se *tab* ≠ *tab_Anc*.

, se *E* está contido em um indicador de ordem *sequence* em um elemento ancestral.

Exemplo:

```
<xs:element name="Ea" type="anyType"/>
```

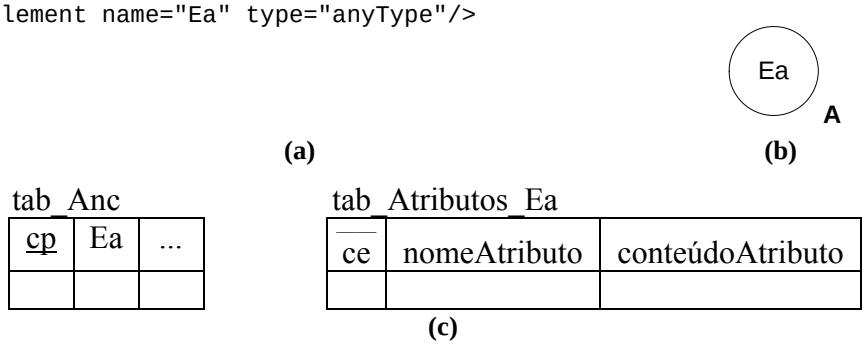


Figura 5-21. Exemplo de Mapeamento de Elemento anyType: (a) esquema XML; (b) representação do elemento no modelo XML Schema; (c) esquema relacional.

[7] Um *Ea* com *n* atributos sempre gera *n* linhas em *tab_Atributos_EA*, uma linha para cada atributo de *Ea*.

5.5. Algoritmo de Conversão

Após a realização das transformações do pré-processamento, é necessário aplicar as regras de mapeamento sobre as estruturas do esquema XML. A aplicação destas regras é feita na forma de um algoritmo de conversão, que define a seqüência de aplicação das mesmas.

A definição em alto nível do algoritmo encontra-se a seguir.

```
-------------------------
Algoritmo de Conversão(esquema XML Schema XSD)
Inicio
     Para cada elemento global Ek ∈ XSD faça
     TrataElemento(Ek);
Fim.


Procedimento TrataElemento(Ek)
Início
```

- Identifique o tipo de *Ek* e aplique a regra correspondente ao seu tipo;
- Se *Ek* possui um conjunto de atributos *Ca* então
  Para cada *ai* ∈ *Ca* faça
  Identifique o tipo de *ai* e aplique a regra correspondente ao seu tipo;
- Se *Ek* possui um conjunto de sub-elementos *Ce* então
  Para cada *ei* ∈ *Ce* faça TrataElemento(ei);

```
Fim.
```

O algoritmo é recursivo e trabalha da seguinte forma: o `Algoritmo de Conversão` recebe como entrada um arquivo XSD pré-processado que será mapeado. Para cada declaração de elemento global encontrado no XSD, o algoritmo executa o procedimento recursivo `TrataElemento` passando como parâmetro este elemento global.

O procedimento `TrataElemento`, por sua vez, identifica o tipo do elemento que recebeu como parâmetro e aplica a regra de mapeamento correspondente. Verifica ainda se o elemento possui atributos e em caso positivo identifica o tipo de

cada um destes atributos e aplica as regras de mapeamento de atributos para cada um deles. Depois, verifica se possui sub-elementos, e se tiver, identifica o tipo destes elementos e para cada um deles executa o procedimento `TrataElemento` passando-os como parâmetro para o mesmo, estabelecendo desta maneira a recursão.

No esquema relacional gerado, as colunas que foram mapeadas a partir de elementos ou atributos indicados com opcionais, devem poder assumir valores nulos (padrão para atributos no modelo relacional) e para aqueles indicados como obrigatórios, as colunas não podem assumir valores nulos.

6. Estudo de Caso

Para exemplificar a abordagem de mapeamento de um esquema em XML Schema para um esquema relacional definida neste trabalho, foi escolhido um domínio de aplicação a respeito de uma universidade.

6.1.1. Requisitos do Domínio

Esta *Universidade* possui *Centros* que são identificados por *siglas,* Os Centros possuem *Departamentos* e *Observações*. Estes Departamentos possuem *Cursos* (no mínimo um (01)), *Observações* e *Laboratórios*. Os *Cursos* oferecidos nesta universidade possuem um *Nome* e podem ser de dois tipos: *Graduação* ou *Pós-Graduação;* possuem *Disciplinas* e *Declarações de Matrícula* que apresentam o *Nome do Aluno* e a *Data*. As disciplinas do curso possuem *Ementas* e *Turmas.* As turmas possuem até quarenta (40) *Alunos* (cada aluno possui *Nome*, *Sobrenome*, *sexo* e *registro acadêmico*) e um (1) *Professor* (cada professor possui *Nome*, *Sobrenome*, *sexo* e *CPF*). Um professor pode ser um *Professor Visitante*, possuindo adicionalmente uma *instituição de origem*.

A Figura 6-1 (a) apresenta o esquema definido em XML Schema que descreve esta Universidade, levando em consideração os requisitos descritos anteriormente. A Figura 6-1 (b) mostra a representação gráfica no modelo XML Schema equivalente a este esquema.

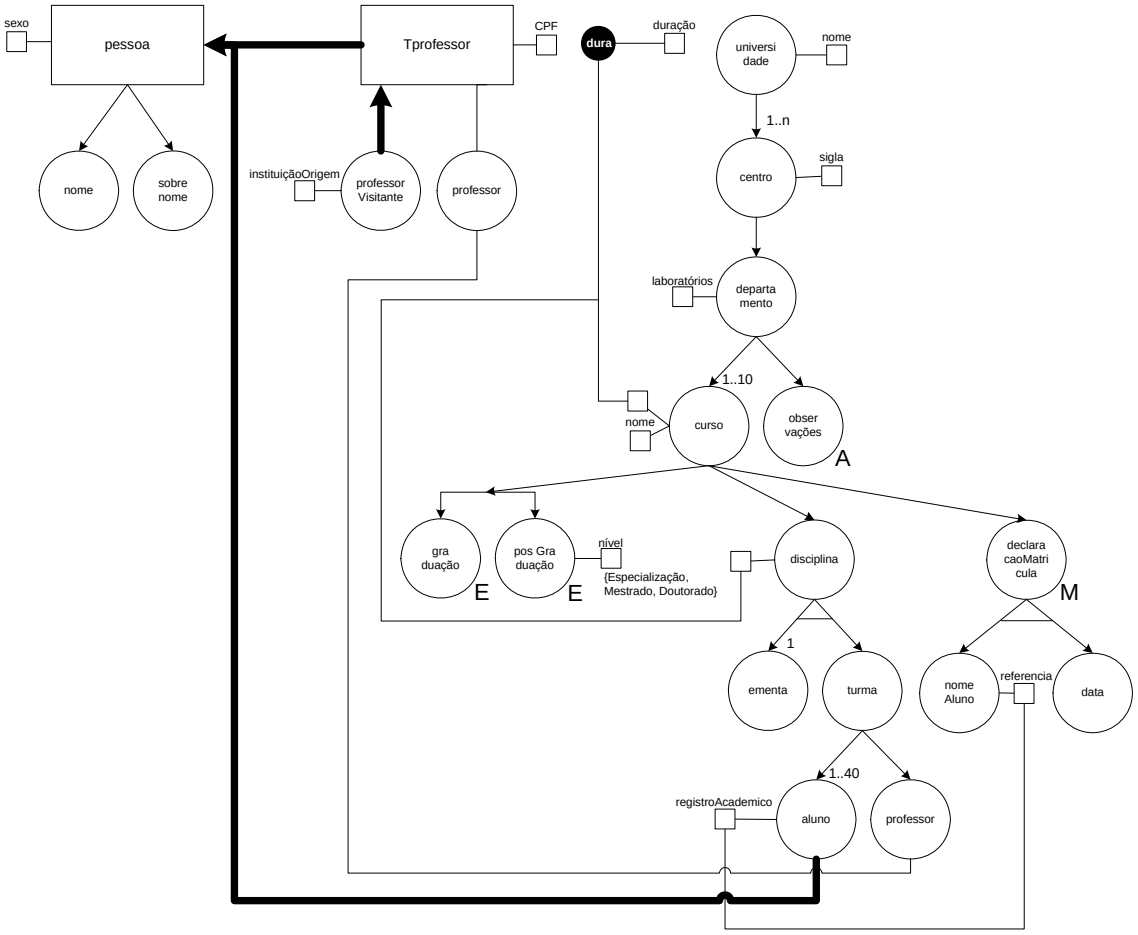
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Arquivo do esquema XML de Entrada-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">

<xs:complexType name="pessoa" abstract="true">
  <xs:sequence>
   <xs:element name="nome" type="xs:string"/>
   <xs:element name="sobrenome">
    <xs:simpleType>
     <xs:restriction base="xs:string">
       <xs:maxLength value="20"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
  </xs:sequence>
```

```
  <xs:attribute name="sexo" type="xs:string"/>
</xs:complexType>

<xs:complexType name="Tprofessor">
 <xs:complexContent>
  <xs:extension base="pessoa">
  <xs:attribute name="CPF">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:pattern value="[0-9]{9}-[0-9]{2}"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  </xs:extension>
 </xs:complexContent>
 </xs:complexType>

<xs:element name="professor" type="Tprofessor"/>

<xs:element name="professorVisitante" substitutionGroup="professor">
<xs:complexType>
 <xs:complexContent>
  <xs:extension base="Tprofessor">
  <xs:attribute name="insituiçãoOrigem" type="xs:string"/>
  </xs:extension>
 </xs:complexContent>
 </xs:complexType>
</xs:element>

<xs:attributeGroup name="dura">
 <xs:attribute name="duração">
  <xs:simpleType>
   <xs:union memberTypes="xs:string xs:int"/>
  </xs:simpleType>
 </xs:attribute>
</xs:attributeGroup>

<xs:element name="universidade">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="centro" minOccurs="1">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="departamento">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="curso" maxOccurs="10">
          <xs:complexType>
           <xs:sequence>
            <xs:choice>
             <xs:element name="graduacao">
              <xs:complexType></xs:complexType>
             </xs:element>
             <xs:element name="posgraduacao">
              <xs:complexType>
               <xs:attribute name="nível">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                  <xs:enumeration value="Especialização"/>
```

```
                <xs:enumeration value="Mestrado"/>
                <xs:enumeration value="Doutorado"/>
                </xs:restriction>
               </xs:simpleType>
              </xs:attribute>
             </xs:complexType>
            </xs:element>
           </xs:choice>
           <xs:element name="disciplina">
            <xs:complexType>
             <xs:all>
              <xs:element name="ementa" type="xs:string" minOccurs="1"/>
              <xs:element name="turma">
               <xs:complexType>
                <xs:sequence>
                 <xs:element name="aluno" maxOccurs="40">
                  <xs:complexType>
                   <xs:complexContent>
                    <xs:extension base="pessoa">
                     <xs:attribute name="registroAcademico" type="xs:ID"
use="required"/>
                    </xs:extension>
                   </xs:complexContent>
                  </xs:complexType>
                 </xs:element>
                 <xs:element ref="professor"/>
                </xs:sequence>
                </xs:complexType>
               </xs:element>
             </xs:all>
             <xs:attributeGroup ref="dura"/>
             </xs:complexType>
           </xs:element>
           <xs:element name="declaracaoMatricula">
            <xs:complexType mixed="true">
             <xs:all>
              <xs:element name="nomeAluno">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="referencia" type="xs:IDREF"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             <xs:element name="data" type="xs:date"/>
             </xs:all>
             </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attributeGroup ref="dura"/>
          <xs:attribute name="nome" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="observações" type="xs:anyType"/>
       </xs:sequence>
       <xs:attribute name="laboratórios">
        <xs:simpleType>
         <xs:list itemType="xs:string"/>
```

```
        </xs:simpleType>
       </xs:attribute>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
    <xs:attribute name="sigla" type="xs:string" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="nome" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

(a)



(b)

Figura 6-1. (a)Esquema em XML Schema para Universidade (b) representação do esquema no modelo XML Schema.

6.1.2. Pré-Processamento do Esquema de entrada

A primeira etapa da abordagem de mapeamento é o pré-processamento. A Figura 6-2 mostra o resultado da aplicação desta etapa sobre o esquema XML da Figura 6-1.

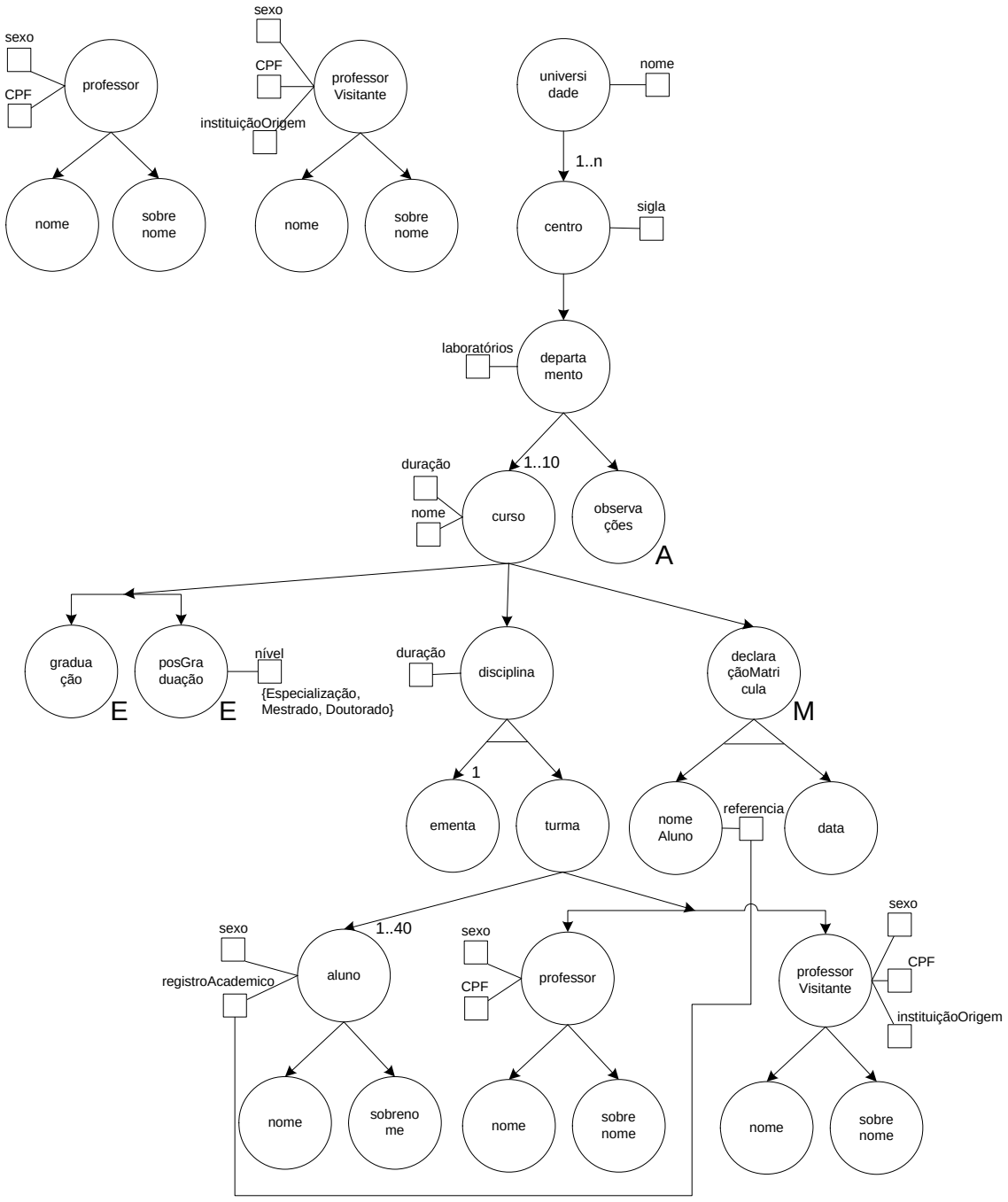
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Arquivo Resultante da aplicação do Pré-processamento-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">

<xs:element name="professor">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="nome" type="xs:string"/>
   <xs:element name="sobrenome">
    <xs:simpleType>
     <xs:restriction base="xs:string">
       <xs:maxLength value="20"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="sexo" type="xs:string"/>
  <xs:attribute name="CPF">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:pattern value="[0-9]{9}-[0-9]{2}"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
</xs:element>

<xs:element name="professorVisitante">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="nome" type="xs:string"/>
   <xs:element name="sobrenome">
    <xs:simpleType>
     <xs:restriction base="xs:string">
       <xs:maxLength value="20"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="sexo" type="xs:string"/>
  <xs:attribute name="CPF">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:pattern value="[0-9]{9}-[0-9]{2}"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
```

```
  <xs:attribute name="insituiçãoOrigem" type="xs:string"/>
 </xs:complexType>
</xs:element>

<xs:element name="universidade">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="centro">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="departamento" minOccurs="1">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="curso" maxOccurs="10">
          <xs:complexType>
           <xs:sequence>
            <xs:choice>
             <xs:element name="graduacao">
              <xs:complexType></xs:complexType>
             </xs:element>
             <xs:element name="posgraduacao">
              <xs:complexType>
               <xs:attribute name="nível">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                  <xs:enumeration value="Especialização"/>
                  <xs:enumeration value="Mestrado"/>
                  <xs:enumeration value="Doutorado"/>
                  </xs:restriction>
                 </xs:simpleType>
                </xs:attribute>
              </xs:complexType>
             </xs:element>
            </xs:choice>
             <xs:element name="disciplina">
             <xs:complexType>
              <xs:all>
               <xs:element name="ementa" type="xs:string" minOccurs="1"/>
               <xs:element name="turma">
                <xs:complexType>
                 <xs:sequence>
                  <xs:element name="aluno" maxOccurs="40">
                  <xs:complexType>
                    <xs:sequence>
                     <xs:element name="nome" type="xs:string"/>
                     <xs:element name="sobrenome">
                      <xs:simpleType>
                       <xs:restriction base="xs:string">
                         <xs:maxLength value="20"/>
                       </xs:restriction>
                      </xs:simpleType>
                     </xs:element>
                    </xs:sequence>
                    <xs:attribute name="sexo" type="xs:string"/>
                    <xs:attribute name="registroAcademico" type="xs:ID"
use="required"/>
                 </xs:complexType>
                 </xs:element>
                 <xs:choice>
```

```
                <xs:element name="professor">
                 <xs:complexType>
                  <xs:sequence>
                   <xs:element name="nome" type="xs:string"/>
                   <xs:element name="sobrenome">
                    <xs:simpleType>
                     <xs:restriction base="xs:string">
                       <xs:maxLength value="20"/>
                     </xs:restriction>
                    </xs:simpleType>
                   </xs:element>
                  </xs:sequence>
                  <xs:attribute name="sexo" type="xs:string"/>
                  <xs:attribute name="CPF">
                   <xs:simpleType>
                    <xs:restriction base="xs:string">
                     <xs:pattern value="[0-9]{9}-[0-9]{2}"/>
                    </xs:restriction>
                   </xs:simpleType>
                  </xs:attribute>
                 </xs:complexType>
                </xs:element>
                <xs:element name="professorVisitante">
                <xs:complexType>
                 <xs:sequence>
                  <xs:element name="nome" type="xs:string"/>
                  <xs:element name="sobrenome">
                   <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:maxLength value="20"/>
                    </xs:restriction>
                   </xs:simpleType>
                  </xs:element>
                 </xs:sequence>
                 <xs:attribute name="sexo" type="xs:string"/>
                 <xs:attribute name="CPF">
                  <xs:simpleType>
                   <xs:restriction base="xs:string">
                    <xs:pattern value="[0-9]{9}-[0-9]{2}"/>
                   </xs:restriction>
                  </xs:simpleType>
                 </xs:attribute>
                 <xs:attribute name="insituiçãoOrigem"
type="xs:string"/>
                </xs:complexType>
               </xs:element>
               </xs:choice>
              </xs:sequence>
              </xs:complexType>
             </xs:element>
            </xs:all>
            <xs:attribute name="duração">
             <xs:simpleType>
              <xs:union memberTypes="xs:string xs:int"/>
             </xs:simpleType>
            </xs:attribute>
            </xs:complexType>
          </xs:element>
          <xs:element name="declaracaoMatricula">
```

```
           <xs:complexType mixed="true">
            <xs:all>
             <xs:element name="nomeAluno">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:string">
                 <xs:attribute name="referencia" type="xs:IDREF"/>
                </xs:extension>
               </xs:simpleContent>
              </xs:complexType>
             </xs:element>
            <xs:element name="data" type="xs:date"/>
            </xs:all>
            </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="duração">
          <xs:simpleType>
           <xs:union memberTypes="xs:string xs:int"/>
          </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="nome" type="xs:string" use="required"/>
         </xs:complexType>
       </xs:element>
       <xs:element name="observações" type="xs:anyType"/>
      </xs:sequence>
      <xs:attribute name="laboratórios">
       <xs:simpleType>
        <xs:list itemType="xs:string"/>
       </xs:simpleType>
      </xs:attribute>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="sigla" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:sequence>
<xs:attribute name="nome" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

(a)



(b)

Figura 6-2. (a)Esquema em XML Schema para Universidade Pré-Processado; (b) representação do esquema no modelo de XML Schema..

Para facilitar o entendimento da aplicação da etapa de pré-processamento, algumas das alterações realizadas no esquema XML de entrada são comentadas a seguir:

- O grupo de substituição, gerado pelo elemento *professorVisitante*, que referencia o elemento *professor*, através do atributo *substitutionGroup*,

(definindo-o, desta maneira, como cabeça do grupo de substiuição), é alterado incluindo um indicador de ordem *choice* no elemeno turma, que faz uso do elemento professor; este *choice* contém o elemento *professor* e *professorVisitante*, permitindo o uso de um dos dois (idêntido à maneira como o grupo de substituição havia definido);

- O elemento *aluno* estende a definição do tipo *pessoa*, que contém os elementos *nome* e *sobrenome* e um atributo *sexo*, incluindo o atributo *registroAcadêmico*. A alteração relizada incorpora a declaração dos elementos *nome* e *sobrenome* e do atributo *sexo,* antes realizada em *pessoa,* dentro do elemento *aluno*, fazendo com que estas declarações sejam locais no elemento *aluno.* Outro exemplo de tratamento de extensões pode ser visto quando o elemento *professorVisitante* passa a incorporar localmente o conteúdo do tipo *Tprofessor*;
- A eliminação de referências pode ser vista quando a referência ao elemento *professor*, feita pelo elemento *turma* com o uso de *ref*, é substituída pela cópia da declaração do elemento *professor*. Pode ser vista também na declaração do elemento *professor* que faz referência ao tipo *Tprofessor* com o atributo *type*; após a substituição, o conteúdo deste tipo é incorporado localmente no elemento *professor*. Além destes, os elementos *curso* e *disciplina* fazem referência ao grupo de atributos *dura*, que contém um atributo *duração*. As alterações realizadas para eliminar estas referências incluem o atributo *duração* na definição de *curso* e *disciplina*;
- Após a eliminação das referências existentes no esquema, o grupo de atributos *dura* e os tipos de dados *pessoa* e *Tprofessor* foram removidos do esquema, como parte do processo de elminação de referências.

As instâncias que podem ser geradas a partir do esquema XSD pré-processado também são instâncias válidas do esquema XSD original; isto mostra que o pré-processamento não altera a semântica do esquema.

6.1.3. Aplicação do Algoritmo de Conversão

A execução do algoritmo de conversão produz como resultado o esquema relacional. Alguns exemplos da aplicação deste algoritmo sobre conceitos do modelo XML Schema que descreve a Universidade são comentados a seguir.

Incialmente, o `AlgoritmoConversão` recebe o arquivo XSD pré-processado, apresentado na Figura 6-2. Seguindo os passos do algoritmo, todos os elementos globais do esquema são investigados. Para o elemento *professor*, por exemplo, o procedimento `TrataElemento` é executado recebendo este elemento como parâmetro. O elemento *professor* tem seu tipo identificado como elemento complexo. Portanto, a **regra para mapeamento de elemento complexo** é aplicada sobre o mesmo. Como resultado desta aplicação, a tabela *tab_professor* é gerada, incluindo uma coluna chave primária *cp.*

O elemento *professor* possui um sub-elemento *nome*. O procedimento `TrataElemento` é executado novamente para *nome*, de maneira recursiva. Nesta execução o elemento *nome* é identificado como elemento de tipo simples contido em um ancestral com indicador de ordem *sequence*, e após a aplicação da **regra de mapeamento para elemento simples de tipo nativo**, são geradas: uma coluna *nome* que guarda o conteúdo do elemento; e uma coluna *ordem_nome* que mantém a ordem proposta pelo indicador *sequence*, ambas em *tab_professor*. Terminado o procedimento, a recursão retorna e encontra o elemento *sobrenome* que é tratado da mesma maneira com a aplicação da **regra de mapeamento para elemento simples derivado por restrição**, gerando uma estrutura similar com a criação de restrições de integridade indicando que a coluna *sobrenome* possui tamanho vinte (20).

No elemento *professor*, existem ainda os atributos *sexo* e *CPF*. Aplicam-se, respectivamente, a **regra para mapeamento de atributo de tipo nativo** e a **regra para mapeamento de atributo de tipo derivado por restrição**. Para o atributo *sexo* é gerada uma coluna de nome *sexo* e para o atributo *CPF* é gerada uma coluna de nome *CPF*, ambas na tabela *tab_professor*.

Os elementos graduação e posgraduação são identificados pelo algoritmo e tratados pela **regra para mapeamento de elemento complexo vazio**, gerando respectivamente uma coluna *graduação* em *tab_departamento_curso* e uma tabela

tab_posgraduação com uma coluna chave estrangeira *ce*. O atributo *nível,* do elemento vazio *posgraduação* é mapeados para uma coluna nível em *tab_posgraduação.*

O atributo *duração* do elemento *curso* é mapeado pela **regra para mapeamento de atributo de tipo *Union,*** que gera uma coluna *duração* de tipo *string*(tipo mais abrangente da *Union*), na tabela *tab_departamento_curso*.

O elemento *observações* contido em *departamento*, identificado pelo algoritmo com tipo anyType, é mapeado com a **regra para mapeamento de elemento de tipo anyType**. Este mapeamento gera uma tabela *tab_departamento_observações*, e gera também uma tabela *tab_Atributos_observações* que armazena os atributos de *observações*, que podem ser de qualquer tipo e quantidade visto que o elemento é de tipo *anyType*.

A **regra para mapeamento de elemento complexo de tipo misto**, é aplicada sobre o elemento declaraçãoMatrícula. É gerado no modelo relacional uma tabela tab_curso_declaraçãoMatrícula com chave primária *cp*, chave estrangeira *ce* e uma coluna *declaraçãoMatricula*.

O atributo *laboratórios* pertencente ao elemento *departamento*, é identificado pelo procedimento como atributo de tipo list, que então aplica a **regra para mapeamento de atributo de tipo list**. Este mapeamento gera uma tabela de nome *tab_departamento_laboratórios* com coluna *ce*, que referencia *tab_centro_departamento*, e uma coluna *laboratórios* que contém um item da lista.

Os demais elementos do esquema, foram tratados de maneira semelhante, com a aplicação das regras correspondentes aos seus tipos pelo procedimento `TrataElemento` que foi executado pelo algoritmo para cada elemento global.

O esquema relacional resultante da execução do `AlgoritmoConversão` pode ser visto na Figura 6-3. No caso da existência de restrições de integridade, o anexo é apresentado ao lado, ou abaixo, da tabela correpondente.

tab_professor

cp	sexo	CPF	nome	ordem_nome	sobrenome	ordem_sobrenome

- O valor de CPF deve seguir o formato [0-9]{9}-[0-9]{2}
- o comprimento da coluna sobrenome é 20

tab_professorVisitante

cp	sexo	CPF	instituiçãoOrigem	nome	ordem_nome	sobrenome	ordem_sobrenome

- O valor de CPF deve seguir o formato [0-9]{9}-[0-9]{2}
- o comprimento da coluna sobrenome é 20

tab_universidade_nome

cp	nome

a coluna nome não pode assumir valores nulos

tab_universidade_centro

cp	ce	sigla	ordem_centro

tab_centro_departamento

cp	ce	ordem_departamento

departamento só possui uma ocorrência em centro

tab_departamento_laboratórios

ce	laboratórios

tab_departamento_curso

cp	ce	duração	nome	ordem_curso	graduação

- curso pode ter até 10 ocorrências em departamento
- a ocorrência da coluna *graduação* deve ser disjunta em relação a valores na tabela tab_curso_posGraduação, devido ao choice

tab_departamento_observações

cp	ce	observações	ordem_observações

tab_Atributos_observações

ce	nomeAtributo	conteudoAtributo

tab_curso_posGraduação

ce	nível

- o conteúdo da coluna nível pode ser “Especialização”, “Mestrado” ou “Doutorado”.
- a ocorrencia de valores nesta tabela deve ser disjunta em relação à coluna *graduação* de *tab_departamento_curso*, devido ao *choice*.

tab_curso_disciplina

cp	ce	duração	ordem_disciplina	ementa

a coluna ementa não pode assumir valor nulo.

tab_disciplina_turma

cp	ce

tab_turma_aluno

cp	ce	sexo	registroAcadêmico	ordem_aluno	nome	ordem_nome	sobrenome	ordem_sobrenome

- pode haver até 40 ocorrências de aluno
- o comprimento da coluna sobrenome é 20

tab_turma_professor

cp	ce	sexo	CPF	ordem_professor	nome	ordem_nome	sobrenome	ordem_sobrenome

- a ocorrência em tab_turma_professor e tab_turma_professorVisitante deve ser disjunta devido ao choice
- o comprimento da coluna sobrenome é 20

tab_turma_professorVisitante

cp	ce	sexo	CPF	instituiçãoOrigem	ordem_professorVisitante	nome	ordem_nome	sobrenome	ordem_sobrenome

- o valor de CPF deve seguir o formato [0-9]{9}-[0-9]{2}
- a ocorrência em tab_turma_professor e tab_turma_professorVisitante deve ser disjunta devido ao choice
- o comprimento da coluna sobrenome é 20

tab_curso_declaraçãoMatrícula

cp	ce	declaraçãoMatricula	data

tab_declaraçãoMatrícula_nomeAluno

ce	nomeAluno	referencia

Figura 6-3. Esquema relacional resultante do mapeamento do XML Schema da universidade.

7. CONCLUSÃO

O uso de documentos XML para integração de sistemas, assim como o uso de esquemas XML Schema para controlar e validar os documentos XML, cresceu consideravelmente desde a criação destas tecnologias e hoje é uma realidade. Como estes processos de integração geralmente envolvem bancos de dados relacionais, torna-se necessário criar mecanismos para viabilizar a conversão de dados XML, descritos em XML Schema, para o modelo relacional. Portanto, a abordagem proposta por esta dissertação é relevante neste contexto.

Pode-se perceber o grande interesse nas tecnologias XML e XML Schema, dado o grande número de linguagens de marcação XML que surgiram para atender necessidades de padronização do conteúdo de dados em diversas áreas de aplicação, como o padrão OTP–Open Trading Protocol, usado em Comércio Eletrônico, MML–Mathematical Markup Language, usado para definição de expressões matemáticas, SVG –Scalable Vector Graphics, para Gráficos Vetoriais e GML – Geographic Markup Language, usada na área de Geoprocessamento.

As regras de mapeamento propostas são bastante detalhadas, pois contemplam grande parte dos conceitos do modelo XML Schema, se comparadas com trabalhos relacionados, que não consideram tantos conceitos como os analisados nesta dissertação. De maneira geral, estes trabalhos não permitem que o desenvolvedor empregue sua experiência para otimizar o modelo resultante da transformação, já que o resultado do mapeamento no sentido XML→relacional, gera especificações em SQL em alguns casos. Além deste problema em alguns trabalhos os conceitos do XML Schema são considerados parcialmente. A apresentação das regras de mapeamento muitas vezes se dá de maneira implícita, embutida no código do sistema proposto.

Ainda, a abordagem de mapeamento proposta é **genérica**, visto que atua no nível lógico relacional, permitindo ao desenvolvedor aplicar seus conhecimentos para obter um modelo físico mais otimizado para o seu SGBD relacional. A abordagem é também considerada **flexível**, pois propõe, em diversas regras, várias opções para o mapeamento de um conceito do modelo XML Schema.

Durante o desenvolvimento desta dissertação notou-se ainda algumas situações relevantes para pesquisa, mas que por estarem fora do escopo da mesma, são listadas a seguir, abrindo novas possibilidades de investigação, ampliando a possibilidade de aplicação e melhorando ainda mais os resultados obtidos.

- *Incorporar os conceitos de key e keyref.* Analisar maneiras de mapear as definições de *key e keyref*, sobre elementos e atributos, no esquema relacional resultante. O uso de *key* e *keyref* produz resultados semelhantes ao uso de *ID* e *IDREF*, as principais diferenças são: *key* e *keyref* (i) atuam a nível local e (ii) permitem definir conteúdos únicos envolvendo elementos e atributos, ao contrário de *ID* e *IDREF* que atuam em nível global nas instâncias e se aplicam somente a atributos. E, por este motivo, não foram contemplados neste trabalho;
- *Estudar situações de reestruturação do esquema relacional*, com o objetivo de tornar o esquema relacional mais otimizado. Esta pode ser uma etapa posterior à execução do algoritmo de conversão. Uma possível reestruturação do esquema seria o *particionamento de tabelas*. Em casos onde, no esquema relacional resultante, um conjunto de dados se repete em mais de uma tabela, é possível agrupar este conjunto de dados em uma tabela separada que é referenciada pelas tabelas que continham este conjunto. O objetivo de tal separação seria a eliminação de redundâncias;
- *Definir um processo de mapeamento de instâncias de dados*. Este processo forneceria alternativas que seriam escolhidas, para o fracionamento e armazenamento dos dados de uma instância XML no esquema relacional gerado;
- *Gerar regras de mapeamento para o modelo Objeto-Relacional.* O modelo objeto-relacional possui características como estruturas aninhadas, atributos multivalorados, listas de valores, que possuem uma relação mais direta com algumas características do modelo XML Schema. Um estudo destas características para analisar a possibilidade de definição de regras de mapeamento que explorem estas relações para facilitar a conversão entre os modelos.

Com isto conclui-se que este trabalho produziu contribuições para a pesquisa na área de XML e banco de dados, tendo alcançado os objetivos iniciais.

8. BIBLIOGRAFIA

[AME03] Amer-Yahia, Sihem. **Storage techniques and mapping schemas for XML**. Technical Report TD-5P4L7B, AT&T Labs-Research, Maio 2003.

[AME04] Amer-Yahia, Sihem; Du, Fang; Freire, Juliana. **A Generic and Flexible Framework for Mapping XML Documents into Relations**. WWW2004, Maio 17-22, New York, 2004.

[BRA04] Bradley, N.: **The XML Companion**. Harlow, USA: Addison-Wesley Longman Limited, 2004.

[COD70] Codd, E. F. **A relational model of data for large shared data banks**. Communications of the ACM, Volume 13 Issue 6. June 1970

[COD80] Codd, E. F. **Data models in database management**. Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling, Volume 11 ,16 Issue 2. June 1980

[CHE76] Chen, P.P., **The Entity-Relationship Model: Toward a unified view of data**, ACM Transactions on Database Systems, v.1, n.1, 1-36, 1976.

[DUT04]Duta, Angela Cristina; Barker, Ken; Alhajj, Reda. **ConvRel: Relationship Conversion to XML Nested Structures**. Symposium on Applied Computing 2004: 698-702.

[ELM02] Elmasri , Ramez; Navathe, Shamkant B. **Sistemas de Banco de Dados - Fundamentos e Aplicações**. Editora LTC, Terceira Edição, 2002

[FER00] Fernandez M., Tan W.C., D. Suciu. **SilkRoute: Trading between Relations and XML**. WWW9/Computer Networks 33(1-6):723-745. 2000

[FLO04]Flora Dilys Salim, Rosanne Price, Maria Indrawan, Shonali Krishnaswamy. **Graphical Representation of XML Schema**. APWeb 2004: 234-245

[FON03] Fong, J.; Wong, H.K.; Cheng, Z.. **Converting relational database into XML documents with DOM**. Information and Software Technology, 45: 335-355. Elsevier. 2003

[FRA04] François Yergeau, 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/

[HAN03] HAN, Wook-Shin, et al; **An XML Storage System for Object-Oriented/ Object-Relational DBMSs**. JOURNAL OF OBJECT TECHNOLOGY. Vol. 2, No. 3, May-June 2003

[KAP04]Kappel, Gerti; Kapsammer, Elisabeth; Retschitzegger, Werner. **Integrating XML and Relational Database Systems**. World Wide Web: Internet and Web Information Systems, 7, 343–384, 2004

[MAR01] MARTIN, Didier; et al. **Professional XML**. Rio de Janeiro: Editora Ciência Moderna Ltda, 2001. 1º. 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.

[MUR03] Murthy, Ravi; Banerjee, Sandeepan. **XML Schemas in Oracle XML DB**. In: Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[SIL99] Silberschatz, Abraham; Korth, Henry F.; Sudarshan, S. **Sistema de Banco de Dados**. Terceira Edição, Makron Books, 1999.

[VAR04] Varlamis, Iraklis; Vazirgiannis, Michalis. **Bridging XML-Schema and relational databases. A system for generating and manipulating relational databases using valid XML documents**. Proceedings of the ACM Symposium on Document Engineering, 2001. 105-114.

[W3C04] W3C Consortium. **XML Schema**. Disponível em http://www.w3.org/XML/Schema

ANEXO 1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E DE ESTATÍSTICA
GRUPO DE BANCO DE DADOS

MODELO XML SCHEMA
CARACTERÍSTICAS E REPRESENTAÇÃO GRÁFICA

RELATÓRIO DE PESQUISA DO GRUPO DE BANCO DE DADOS
NÚMERO 1

Claudio R. M. Mauricio
Fabiana F. F.Peres
Ronaldo dos Santos Mello

FLORIANÓPOLIS, AGOSTO DE 2005

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho foi desenvolvido no contexto do grupo de Banco de Dados da Universidade Federal de Santa Catarina para ser utilizado como referência comum nas demais atividades do grupo que envolvem XML e XML Schema. Tem por objetivo apresentar os recursos disponibilizados pela linguagem XML Schema, de forma acessível, tendo por base as normativas da W3C que contém sua definição.Além disso apresenta uma proposta de modelo de representação gráfica criado para facilitar a compreensão de estruturas XML Schema descrevendo visualmente a organização estrutural de dados XML.

Palavras Chaves: XML Schema; modelo de documentos; representação gráfica.

1. XML e XML Schema

XML (eXtensible Markup Language) [YER04] é uma linguagem de marcação, recomendada pela W3C (*World Wide Web Consortium*), para descrição de informações. Sua especificação está disponível no site http://www.w3.org/XML/.

XML está estruturada de maneira a garantir flexibilidade e robustez na descrição de dados pois não possui elementos pré-definidos, é o usuário quem define o conjunto de elementos que melhor se aplica no contexto para representação de seus dados . Por isso é considerada uma meta-linguagem.

Um documento XML apresenta uma estrutura em forma de árvore hierárquica simples contendo marcações e dados. As marcações XML são construídas por elementos. Cada elemento XML é delimitado por uma *tag inicial* (`<nome>`) e uma *tag final* (`</nome>`). O seu conteúdo pode ser outros elementos, dados, e/ou outras marcações [MAR01].

Os elementos XML, podem ainda ter atributos associados a eles. Os atributos são compostos de um par nome-valor especificados na *tag* de início do elemento (`<pessoa rg=“5.191.192-3”>`).

A Figura 1-1 apresenta um documento XML que exemplifica o uso de elementos contendo outros elementos e/ou dados e também atributos associados a elementos.

```
<?xml version="1.0" encoding="ASCII" standalone="yes"?>
 <pessoa 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>Floranópolis</cidade>
   </endereço>
  </pessoa>
```

Figura 1-1. Documento XML.

A motivação inicial para a definição da XML foi a necessidade de intercâmbio de informações na *Web* de maneira independente. Por ser um padrão aberto, permite a comunicação entre aplicações diferentes e sua natureza auto-descritiva a torna uma opção eficaz para solucionar problemas de B2B (*Business to Business*) e extranet.

Como dito anteriormente, um documento XML é composto por um conjunto qualquer de marcações e dados sobre um contexto. Porém, dois ou mais documentos XML que tem por objetivo descrever informações sobre um mesmo contexto, não necessariamente possuem uma mesma estrutura visto que as marcações do documento não são pré-definidas pela linguagem XML. No entanto, existem situações onde é necessário definir uma estrutura única para ser utilizada pelos documentos XML, como por exemplo, quando estes documentos são utilizados como meio de comunicação entre aplicações.

A estrutura utilizada por um documento XML, pode ser definida pelo uso da linguagem *XML Schema* (*eXtensible Markup Language Schema*).

XML Schema é uma linguagem de marcação desenvolvida em XML, também recomendada pela W3C. Sua especificação está disponível no site http://www.w3.org/XML/Schema .

O objetivo da XML *Schema* é fornecer mecanismos para definição de esquemas, antes resolvido com o uso de DTDs (*Document Type Definition*) [BRA04], porém DTDs são menos expressivos em termos da semântica dos dados. A definição de esquemas padroniza o formato que um documento XML deve ter quando está em conformidade com o mesmo. Um esquema em *XSD (XML Schema Definition)* define estruturas XML declarando elementos, sua ordem, restrições de conteúdo, atributos, grupos reusáveis, novos tipos de dados, enfim declarando tudo que for necessário para estabelecer o que uma instância de um documento XML, em conformidade com este esquema, pode ou deve ter.

Um documento de definição de esquema possui um elemento principal chamado de *schema* e este pode ter vários elementos filhos. Toda a estruturação do esquema se encontra dentro deste elemento.

Analisando as declarações existentes em um esquema, podemos dividi-las em dois grupos:

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

Todas as declarações do primeiro grupo devem estar antes de qualquer do segundo grupo [BRA04]. O grupo de gerenciamento compreende elementos usados para gerenciar a definição do esquema quando este está distribuído em vários arquivos. Já o grupo de componentes de construção do modelo é composto de especificações de elementos, atributos, tipos de dados e outras informações para a definição do esquema. Portanto este trabalho se concentra neste grupo, visto que a semântica das informações é definida através de seus elementos. A Figura 1-2 apresenta os elementos que compõem estes dois grupos de declarações.

```
<schema>
   <!--GRUPO DE GERENCIAMENTO-->
   <import>…</import >
   <include>…</include >
   <redefine>…</redefine >
   <!—GRUPO DE COMPONENTES DE CONSTRUÇÃO DO MODELO-->
   <notation>…</notation >
   <simpleType>…</simpleType >
   <complexType>…</complexType >
   <attributeGroup>…</attributeGroup >
   <attribute…/>
   <group>…</group >
   <element>…</element>
</schema>
```

Figura 1-2. Grupos de Declarações.

A linguagem *XML Schema* possui um conjunto de elementos pré-definidos para permitir a definição de esquemas. Estes elementos, por sua vez, possuem conjuntos de atributos necessários para detalhar suas características, tais como nome, seu uso (opcional ou obrigatório), valor *default*, entre outras. Nas seções que seguem neste capítulo encontra-se um detalhamento destes recursos da linguagem.

1.1. Especificações de Elementos

O principal propósito de definir esquemas é declarar os elementos e atributos que uma instância de documento XML em conformidade com este esquema pode ter e todas as características pertinentes à mesma [BRA04].

Os elementos que irão compor a definição de um esquema são declarados através do uso do elemento *element*. Este elemento possui entre outros, o atributo *name* para especificar seu nome como mostrado na Figura 1-3.

```
<element name= “Aluno” …/>
<element name=”Professor”> … </element>
```

Figura 1-3. Declaração de elementos.

Os elementos declarados em um esquema podem ser do tipo simples ou complexo. São ditos simples quando armazenam uma unidade de informação de um determinado tipo simples abordado na seção 1.5.1. Neste caso é possível especificar um valor *default* ou *fixed* (fixo).

```
<element name=“Apto” type=“integer” default=“0”/>
```

Figura 1-4. Declaração de um elemento simples.

Os elementos ditos complexos são formados por outros sub-elementos e/ou atributos [W3C04]. A Figura 1-5 mostra a declaração de um elemento complexo.

```
<element name=“Aluno”/>
 <complexType>
  ...
  <attribute name=“Nome” type=“string”/>
  <attribute name=“RA” type=“interger”/>
 </complexType>
</element>
```

Figura 1-5. Declaração de um elemento complexo.

Um elemento complexo pode ser classificado quanto a sua estrutura da seguinte forma:

- *Vazio*: não possui nenhum conteúdo e pode ter atributos (Figura 1-6);
- *Contém somente outros elementos:* possui outros sub-elementos e nenhum atributo (Figura 1-7);
- *Contém somente texto:* possui conteúdo simples e atributos (Figura 1-8);

- *Misto:* contém sub-elementos, texto e atributos. Para definir elementos mistos, o atributo *mixed* do tipo complexo do elemento deve ser definido como *true* (Figura 1-9).

```
<element name=“paragrafo”/>
  <complexType>
  </complexType>
</element>
```

Figura 1-6. Declaração de um elemento vazio.

```
<element name=“endereço”
 <complexType>
  <sequence>
   <element name=“rua” type=“string”/>
   <element name=“numero” type=“integer”/>
   <element name=“cidade” type=“string”/>
   <element name=“CEP” type=“string”/>
   <element name=“estado” type=“string”/>
  </sequence>
 </complexType>
</element>
```

Figura 1-7. Declaração de um elemento que contém somente outros elementos.

```
<element name= “endereço”>
 <complexType>
  <simpleContent>
   <extension base=“string”>
     <attribute name=“tipo” type=“string”/>
  </simpleContent>
 </complexType>
</element>
```

Figura 1-8. Declaração de um elemento que contém somente texto.

```
<element name=“Carta”>
```

```
<complexType mixed="true">
 <sequence>
  <element name="nome" type="string">
  <element name="data" type="date">
  …
 </sequence>
</complexType>
</element>
```

Figura 1-9. Declaração de um elemento misto.

Além de permitir a especificação de elementos do tipo simples ou complexo, XML Schema permite que se definam elementos do tipo *anytype*, ou seja, elementos que não possuem nenhuma estrutura pré-definida, podendo ser instanciados com um conteúdo qualquer (Figura 1-10). Para que isso seja possível, o atributo *type* do elemento deve ser do tipo *anytype*.

```
<element name="qualquerCoisa" type="anyType"/>
```

Figura 1-10. Declaração de um elemento do tipo anytype.

Uma declaração equivalente e mais simplificada omite o atributo *type*, pois por *default* ele se torna *anyType* (Figura 1-11).

```
<element name="qualquerCoisa"/>
```

Figura 1-11. Declaração simplificada de um elemento anytype.

Os elementos declarados em um esquema por *default* são obrigados a aparecer uma vez nas instâncias destes esquemas. Porém, é possível especificar seus limites através do atributo *maxOccurs* e *minOccurs*. O primeiro define o número máximo de vezes que um elemento pode ocorrer e o segundo o número mínimo. Para o *maxOccurs* é possível atribuir um numero inteiro maior ou igual a 1 ou ainda atribuir o valor *unbounded* para estabelecer que seu número máximo de vezes é ilimitado e para o *minOccurs* é possível atribuir um numero inteiro qualquer maior ou igual a 0, pois o número mínimo de vezes que um elemento pode aparecer numa instância é nenhuma. Os valores destes atributos não podem ser conflitantes.

Estes dois atributos são chamados de indicadores de ocorrência e definem a quantidade de vezes que um elemento pode ocorrer numa instância de um esquema. A Figura 1-12 apresenta o uso destes indicadores no elemento *Telefone*.

```
<complexType name="dadosPessoais">
```

```
   <sequence>
      <element name=“Nome” type= “string”/>
      <element    name=“Telefone”    type=    “integer”    minOccurs=“1”
      maxOccurs=“4”/>
   </sequence>
</complexType>
```

Figura 1-12. Declaração de um tipo complexo com elementos utilizando indicadores de ocorrência.

1.2. Definição de atributos

Atributos de elementos do esquema podem ser declarados. O elemento *attribute* permite estas declarações. Ele possui, assim como o elemento *element*, o atributo *name* e *type* para especificar respectivamente seu nome e seu tipo de dado, que no caso de atributos, deve ser simples. Todo atributo é por *default* considerado opcional pela linguagem, porém, pode ser declarado explicitamente como tal. O atributo deste elemento que estabelece seu uso é *use* e ele pode assumir o valor *optional, required* ou *prohibited*. A Figura 1-13 mostra a declaração de um atributo como opcional de forma explicita.

```
<element name= “endereço”>
 <complexType>
  ...
  <attribute name=“Apto” type=“integer” use=“optional”>
  ...
 </complexType>
</element>
```

Figura 1-13. Declaração de um elemento complexo contendo um atributo opcional.

Outra flexibilidade oferecida para atributos pela linguagem é permitir que um valor *default* seja pré-estabelecido. Este valor *default* só faz sentido se o atributo *use* do elemento estiver com o valor *optional*, para que quando na ausência da especificação de qualquer valor, ele assuma o *default* [W3C04]. O atributo

disponibilizado para tal é chamado *default.* A Figura 1-14 mostra um exemplo de declaração de atributo com um valor opcional definido.

É interessante também citar neste contexto um outro atributo disponível usado para estabelecer um valor fixo para o atributo. Este atributo é chamado de *fixed*. A Figura 1-15 mostra a declaração de um atributo cujo valor é sempre Br.

Vale destacar que o atributo declarado não pode ter especificado os dois valores (*default* e *fixed*), somente um deles.

```
<attribute name=“País” type =“string” use =“optional” default=“Br”>
```

Figura 1-14. Declaração de um atributo opcional com valor default.

```
<attribute name=“País” type =“string” fixed=“Br”>
```

Figura 1-15. Declaração de um atributo com valor fixo.

Os elementos e atributos filhos diretos do elemento *schema* são considerados globais. Estes podem ser utilizados por outros elementos e atributos do esquema através de referências aos globais (Figura 1-16) e estes elementos globais também podem originar instâncias de documentos XML onde eles são o elemento raiz.

```
<element name="comment" type="string"/>
<complexType name="PurchaseOrderType">
   <sequence>
      <element name="shipTo" type="USAddress"/>
      <element name="billTo" type="USAddress"/>
      <element ref="comment" minOccurs="0"/>
      <element name="items"  type="Items"/>
   </sequence>
</complexType>
```

Figura 1-16. Declaração de um elemento global e seu uso através de referência por um tipo complexo.

O atributo *ref* contém o nome do elemento ou atributo referenciado. Desta forma, este novo elemento ou atributo não terá os atributos *type* e *name* especificados. Quando uma referência é feita, o novo elemento ou atributo especificado terá o mesmo nome e o mesmo tipo que o referenciado.

1.3. Definição de grupos

Além de declarar elementos e atributos individuais de forma global, a XML Schema permite ainda que grupos de elementos e atributos globais sejam declarados. Quando dois ou mais elementos possuem elementos ou atributos em comum, justifica-se a declaração de grupos destes elementos ou atributos, evitando assim a replicação da definição de componentes. O elemento *group* é usado para declarar um grupo de elementos, como mostra a figura Figura 1-17. Já para declarar um grupo de atributos, é usado o elemento *attributeGroup* (Figura 1-19). Ambos possuem o atributo *name* para especificar o nome do grupo. Quando se define um grupo de elementos é necessário especificar a ordem destes elementos fazendo uso de um dos indicadores de ordem. Estes indicadores são detalhados na seção 1.5.2. Uma vez definido o grupo, o seu uso em um elemento é feito através da referência ao grupo, como mostra a Figura 1-18 e a Figura 1-20.

```
<group name=”Endereço”>
   <all>
      <element name=“Rua” type=“string”/>
      <element name=“Numero” type=“positiveInteger”/>
      <element name=“Bairro” type=”string”/>
      <element name=“Telefone” type=”string”/>
   </all>
</group>
```

Figura 1-17. Declaração de um grupo de elementos.

```
<complexType name=“endResidencial”>
      <sequence>
        <group ref=“Endereço”>
      </sequence>
   </complexType>
```

Figura 1-18. Declaração de um tipo complexo que referencia um grupo de elementos.

```
<attributeGroup name=”attrComuns”>
   <attribute name=”RG” type= “string”/>
   <attribute name=”CPF” type= “string”/>
   <attribute name=”DataNascimento” type= “date”/>
</attributeGroup>
```

Figura 1-19. Declaração de um grupo de atributos.

```
<element name=“Aluno”>
   <complexType>
      <sequence>
          <element name=“EnderecoResidencial” type=“endResidencial”
      </sequence>
      <attributeGroup ref=“attrComuns”>
   </complexType>
</element>
```

Figura 1-20. Declaração de um elemento complexo que referencia o grupo de atributos.

Um outro grupo que pode ser criado é o grupo de substituição. Este grupo é definido com o uso do atributo *substitutionGroup*. O elemento cabeça do grupo é um elemento global sem nenhuma construção adicional, é um elemento qualquer. Para criar um grupo de substituição um outro elemento faz referência ao elemento cabeça com o atributo *substitutionGroup*. No exemplo da Figura 1-21 o elemento cabeça é *EndereçoResidencial* devido à referência feita pelo elemento e *EnderecoComercial* que pode ser usado em substituição ao elemento cabeça em qualquer documento de instância. É importante notar que os elementos pertencentes a um grupo de substituição devem ser do mesmo tipo do elemento cabeça ou então de um tipo derivado deste.

```
<element name=“EnderecoResidencial”>…</element>
<element name=“EnderecoComercial”
                substitutionGroup=“EnderecoResidencial”>
...
</element>
```

Figura 1-21 – Declaração de um grupo de substituição

1.4. Definição de Restrições de Identificação

Quando documentos XML são definidos, pode ser necessário controlar o conteúdo dos elementos e atributos que compõe as instâncias. Para tanto, XML Schema permite que se estabeleça chaves fazendo uso dos elementos *unique* ou *key*. Os dois identificam fragmentos únicos de instâncias de documentos [BRA04]. Ambos possuem o atributo opcional *name* que pode ser usado para nomear o fragmento. Este

nome é usado para gerar mensagens de erro consistentes, pois em caso de erro a condição violada é identificada por este nome [BRA04].

Estes dois elementos possuem as seguintes características:

- *unique*: se um elemento opcional é usado como identificador, ele pode não aparecer na instância já que é opcional;
- *key*: mesmo que o elemento usado como identificador seja opcional, ele tem que aparecer na instância devido ao elemento *key*.

Independente de qual dos dois elementos seja usado, duas tarefas devem ser realizadas:

- Identificar o fragmento com o uso de *key* ou *unique*;
- Isolar o valor ou valores que estão dentro destes fragmentos com o uso de dois elementos: *selector* e *field*.

Os elementos *selector* e *field* contêm expressões *XPath* para selecionar os elementos. A linguagem *Xpath*[1] é uma linguagem de consulta para documentos XML que opera na estrutura lógica abstrata do documento XML [XPA99]. *Selector* é usado para isolar/selecionar elementos ou um conjunto de elementos e *field* é usado para isolar/selecionar elementos de tipo simples ou atributos. Quando usados em conjunto, é possível isolar qualquer fragmento usado como chave.

Existe ainda o elemento *keyref*, que é usado para referenciar uma chave do tipo *key* já existente, desta maneira o dado identificado deve ser único nos dois elementos que fazem uso do *key* e do *keyref* . Esta referência é feita por meio do atributo *refer*.

As duas figuras a seguir exemplificam o uso de *key* e *keyref* para controlar o conteúdo de elementos das instâncias XML. A Figura 1-22 define uma chave onde é estabelecido que os atributos *state* e *plateNumber* do elemento *vehicle* devem ser únicos, já a Figura 1-23 mostra uma referência à chave nomeada como "*regKey*" fazendo com que os conteúdos sejam únicos levando em consideração o conjunto de dados do elemento *vehicle* e *car*.

```
<key name="regKey">
  <selector xpath=".//vehicle"/>
  <field xpath="@state"/>
  <field xpath="@plateNumber"/>
</key>
```

Figura 1-22. Definição de chaves com o uso do elemento key.

[1] A especificação da linguagem *XPath* encontra-se em http://www.w3.org/TR/xpath.

```
<keyref name=”carRef” refer=”regKey”>
  <selector xpath=”.//car”/>
  <field xpath=”@regState”/>
  <field xpath=”@regPlate”/>
</keyref>
```

Figura 1-23. Referência de chaves com o uso do elemento keyref.

1.5. Tipos de Dados

A XML Schema suporta dois tipos de dados: simples e complexos.

Os tipos de dados utilizados em uma definição de esquema podem ser classificados como:

- **Anônimos:** tipos de dados (simples ou complexos) especificados dentro do próprio escopo da estrutura. Estes não possuem nomes associados; daí a classificação de anônimo. Esta forma de declarar tipos de dados é adequada quando a estrutura tem uma formação distinta, não havendo a intenção de ser utilizada por outras estruturas;
- **Globais:** tipos de dados (simples ou complexos) especificados como filhos diretos do elemento *schema*. Quando duas ou mais estruturas possuem características similares ou idênticas é adequado declarar estas em tipos globais a fim de evitar replicação de especificações. Estes tipos globais são identificados por um nome (atributo *name* do mesmo) em sua declaração e quando utilizados pelas estruturas são atribuídos ao atributo *type* dos mesmos o nome do tipo global.

A Figura 1-24 apresenta um XML Schema contendo a declaração de um tipo de dado global (*Items*) e outro local anônimo (*complextype* dentro do elemento *item*).

```
<schema>
...
<complexType name="Items">
  <sequence>
    <element name="item" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <sequence>
```

```
            <element name="quantity">
             ...
            </element>
            ...
          </sequence>
        </complexType>
      </element>
    </sequence>
</complexType>
...
</schema>
```

Figura 1-24. Declaração de tipos globais e anônimos.

1.5.1. Tipos de dados simples

A linguagem disponibiliza um grande conjunto de tipos de dados nativos como *string*, *byte* e *integer*. Além destes, novos tipos de dados simples podem ser criados através de restrições impostas pelo usuário ou então fazendo uma lista ou união de vários deles. As restrições, são chamadas de facetas. As facetas são divididas em dois grupos: as aplicáveis a qualquer tipo de dados e as aplicáveis somente aos tipos "*enumerated*". Os tipos *enumerated* compreendem, dentre os tipos simples, aqueles que são ordenados, ou ainda aqueles onde o conceito de ordem é aplicável. A Tabela 1-1 apresenta os tipos de dados nativos e as facetas aplicáveis a todos os tipos de dados. Já a Tabela 1-2 apresenta os tipos de dados *"enumerated"* e as facetas aplicáveis aos mesmos. Um novo tipo de dado simples é criado fazendo uso do elemento *simpleType* (Figura 1-25).

Tabela 1-1. Facetas aplicáveis a todos os tipos de dados nativos.

Tipos simples	Facetas					
	length	minLength	maxLentgh	pattern	enumeration	whitespace
string	sim	sim	sim	sim	sim	sim
normalizedString	sim	sim	sim	sim	sim	sim
token	sim	sim	sim	sim	sim	sim
byte				sim	sim	sim
unsignedByte				sim	sim	sim
base64Binary	sim	sim	sim	sim	sim	sim
hexBinary	sim	sim	sim	sim	sim	sim
integer				sim	sim	sim

positiveInteger				sim	sim	sim
negativeInteger				sim	sim	sim
nonNegativeInteger				sim	sim	sim
nonPositiveInteger				sim	sim	sim
int				sim	sim	sim
unsignedInt				sim	sim	sim
long				sim	sim	sim
unsignedLong				sim	sim	sim
short				sim	sim	sim
unsignedShort				sim	sim	sim
decimal				sim	sim	sim
float				sim	sim	sim
double				sim	sim	sim
boolean				sim		sim
time				sim	sim	sim
dateTime				sim	sim	sim
duration				sim	sim	sim
date				sim	sim	sim
gMonth				sim	sim	sim
gYear				sim	sim	sim
gYearMonth				sim	sim	sim
gDay				sim	sim	sim
gMonthDay				sim	sim	sim
Name	sim	sim	sim	sim	sim	sim
QName	sim	sim	sim	sim	sim	sim
NCName	sim	sim	sim	sim	sim	sim
anyURI	sim	sim	sim	sim	sim	sim
language	sim	sim	sim	sim	sim	sim
ID	sim	sim	sim	sim	sim	sim
IDREF	sim	sim	sim	sim	sim	sim
IDREFS	sim	sim	sim		sim	sim
ENTITY	sim	sim	sim	sim	sim	sim
ENTITIES	sim	sim	sim		sim	sim
NOTATION	sim	sim	sim	sim	sim	sim
NMTOKEN	sim	sim	sim	sim	sim	sim
NMTOKENS	sim	sim	sim		sim	sim

Fonte: [BRA04]

Tabela 1-2. Facetas aplicáveis aos tipos de dados *enumerated*.

Tipos simples	Facetas					
	maxInclusive	maxExclusive	minInclusive	minExclusive	totalDigits	fractionDigits
Byte	sim	sim	sim	sim	sim	sim
unsignedByte	sim	sim	sim	sim	sim	sim
integer	sim	sim	sim	sim	sim	sim
positiveInteger	sim	sim	sim	sim	sim	sim
negativeInteger	sim	sim	sim	sim	sim	sim
nonNegativeInteger	sim	sim	sim	sim	sim	sim
nonPositiveInteger	sim	sim	sim	sim	sim	sim
Int	sim	sim	sim	sim	sim	sim
UnsignedInt	sim	sim	sim	sim	sim	sim
long	sim	sim	sim	sim	sim	sim
unsignedLong	sim	sim	sim	sim	sim	sim
short	sim	sim	sim	sim	sim	sim
unsignedShort	sim	sim	sim	sim	sim	sim
decimal	sim	sim	sim	sim	sim	sim
float	sim	sim	Sim	sim		
double	sim	sim	Sim	sim		
time	sim	sim	Sim	sim		
dateTime	sim	sim	Sim	sim		
Duration	sim	sim	Sim	sim		
Date	sim	sim	Sim	sim		
gMonth	sim	sim	Sim	sim		
gYear	sim	sim	Sim	sim		
gYearMonth	sim	sim	Sim	sim		
gDay	sim	sim	Sim	sim		
GMonthDay	sim	sim	Sim	sim		

Fonte: [BRA04].

```
<simpleType name=”Nota”>
   …
</simpleType>
```

Figura 1-25. Declaração de um tipo de dado simples.

Como mostra a Tabela 1-1 e a Tabela 1-2, existe um conjunto de 12 facetas aplicáveis aos tipos nativos que podem dar origem a um novo tipo de dado simples. A faceta *length* se aplica quando deseja obter um novo tipo de dado simples com tamanho definido (Figura 1-26).

```
<simpleType name=”UF”>
   <restriction base=”string”>
      <lentgh value=”2” />
   </restriction>
</simpleType>
```

Figura 1-26. Aplicação da faceta lentgh.

A faceta *minLength* e a faceta *maxLength* são utilizadas quando se deseja estabelecer um tamanho mínimo e máximo para o novo tipo de dado simples criado, respectivamente (Figura 1-27).

```
<simpleType name=”Estado”>
   <restriction base=”string”>
      <minLentgh value=”3”/>
      <maxLentgh value=”50”/>
   </restriction>
</simpleType>
```

Figura 1-27. Aplicação da faceta *maxLentgh* e *minLentgh*.

A faceta *pattern* é aplicada quando o novo tipo simples é derivado de um tipo nativo pré-estabelecendo o padrão que o dado deve ter. A Figura 1-28 apresenta a definição de um novo tipo simples que restringe o conteúdo do tipo *string* com o uso da faceta *pattern*. A linguagem utilizada para estabelecer o padrão é baseada em expressões regulares da linguagem PERL [BRA04].

```
<simpleType name=”Estado”>
   <restriction base=”string”>
     <pattern value=”[A-Z]{1}[a-zA-Z]{2,}”/>
   </restriction>
</simpleType>
```

Figura 1-28. Aplicação da faceta *pattern*.

A faceta *enumeration* se aplica quando o novo tipo de dado simples possui um conjunto de possíveis valores (Figura 1-29).

```
<simpleType name=”Serie”>
   <restriction base=”string”>
      <enumeration value= ”1o. ano”/>
```

```
      <enumeration value= ”2o. ano”/>
      <enumeration value= “3o. Ano”/>
    </restriction>
</simpleType>
```

Figura 1-29. Aplicação da faceta *enumeration.*

A faceta *whitespace* é aplicada quando o novo tipo de dado simples deve obedecer a uma determinada regra, quando este apresentar caracteres em branco (ou especiais). Os possíveis valores para esta faceta são:

- *replace*: define que marcadores de quebras de linha, tabulação e final de linha serão substituídos por espaços em branco;
- *collapse*: define que os marcadores de quebra de linha, tabulação, espaços antes e depois do texto serão removidos e substitui finais de linha e múltiplos espaços por espaços em branco;
- *preserve*: define que os caracteres em branco devem ser preservados.

A Figura 1-30 define um tipo simples com base no tipo *string* pelo uso da faceta *whitespace*.

```
<simpleType name=”Cidade”>
   <restriction base=”string”>
     <whitespace value=“replace”/>
   </restriction>
</simpleType>
```

Figura 1-30. Aplicação da faceta *whitespace.*

A faceta *minInclusive* e *maxInclusive* aplica-se quando o novo tipo de dado simples está numa faixa de valores entre um mínimo incluindo ele e um máximo também incluindo ele (Figura 1-31). Caso deseje-se que este novo tipo não inclua o valor mínimo e máximo, faz-se uso das facetas *minExclusive* e *maxExclusive*.

```
<simpleType name=”Nota”>
   <restriction base=”decimal”>
     <minInclusive value= ”0”/>
     <maxInclusive value= ”10”/>
   </restriction>
```

```
</simpleType>
```

Figura 1-31. Aplicação da faceta *maxInclusive.*

A faceta *totalDigits* tem a função de pré-estabelecer o número máximo de dígitos que o novo tipo de dado deve ter. A faceta *fractionDigits* pré-estabelece o número máximo de dígitos depois da vírgula que o novo tipo de dado simples pode ter.A Figura 1-32 mostra a definição de um novo tipo de dados simples através do uso destas duas facetas.

```
<simpleType name=”MediaFinal”>
   <restriction base=”decimal”>
     <totalDigits value= ”4”/>
     <fractionDigitis value= ”2”/>
   </restriction>
</simpleType>
```

Figura 1-32. Aplicação das facetas *totalDigits* e *fractionDigits.*

Como dito anteriormente, novos tipos de dados simples também podem ser criados a partir de uma lista de outros tipos de dados simples (Figura 1-33). A linguagem não suporta criar listas de tipos de dados complexos e nem criar listas de outras listas.

```
<simpleType name=”Notas”>
   <list itemType=”Nota”/>
</simpleType>
```

Figura 1-33. Declaração de uma lista de tamanho ilimitado

Na Figura 1-33, o novo tipo de dado simples criado é uma lista ilimitada de tipo Nota. Para restringir o numero de elementos que esta lista pode ter, ou fazer qualquer outra restrição sobre conteúdo da lista, é necessário criar um novo tipo de dados simples, por restrição deste que é uma lista. As facetas *lentgh, maxLentgh, minLentgh* e *whitespace* são aplicáveis sobre os novos tipos de dados criados a partir de lista, como mostra a Figura 1-34.

```
<simpleType name=”NotasLimitadas”>
   <restriction base=”Notas”
     <length value=”4”/>
   </restriction>
</simpleType>
```

Figura 1-34. Declaração de uma lista com tamanho limitado a 4.

Um novo tipo de dado simples também pode ser criado pela união de outros tipos simples (Figura 1-35). O atributo *memberTypes* do elemento *union* da linguagem contém a lista de todos os tipos que compõem este novo tipo criado. As facetas *enumeration* e *pattern* são aplicáveis sobre estes novos tipos de dados.

```
<simpleType name=”CEPES”>
   <union memberTypes=”UF CEP”/>
</simpleType>
```

Figura 1-35. Declaração de um novo tipo de dado simples através da união de outros dados simples.

Como novos tipos de dados simples podem ser especificados com base em outros já especificados, é possível estabelecer que um determinado tipo de dado simples que foi especificado através das facetas *maxExclusive*, *minExclusive*, *maxInclusive*, *minInclusive*, *totalDigits*, *fractionDigits*, *length*, *minLength*, *maxLength* e *whiteSpace* não possa ser alterado com relação à faceta. Para isso, estas facetas possuem o atributo *fixed* que pode assumir o valor *true* ou *false*. Quando este está com o valor *true*, os novos tipos de dados derivados destes não podem alterar o valor daquela faceta. Por *default*, o valor deste atributo é *false*. A Figura 1-36 declara uma lista de 4 elementos que não pode ter seu numero de elementos alterados por outros tipos que venham derivar deste.

```
<simpleType name=”NotasLimitadas”>
   <restriction base=”Notas”
     <length value=”4” fixed = “true”/>
   </restriction>
</simpleType>
```

Figura 1-36. Declaração de um tipo de dado simples que possui tamanho restrito e fixo.

1.5.2. Tipos de dados complexos

A XML Schema não disponibiliza nenhum tipo de dado complexo nativo. Então, para utilizá-los, primeiro é necessário criá-los. Para criar um tipo de dado complexo é necessário utilizar o elemento *complexType* disponibilizado pela linguagem (Figura 1-37).

```
<complexType name=”Endereço”>
   …
</complexType>
```

Figura 1-37. Declaração de um tipo de dado complexo

A diferença base entre tipos de dados simples e complexos é que os primeiros não permitem marcação XML e nem atributos. Já os últimos podem ter marcações XML e atributos associados, como mostra a Figura 1-38. Portanto, um tipo complexo é composto por um conjunto de declarações de elementos e atributos.

```
<complexType name=”Endereço”>
  <sequence>
   <element name=”Rua” type= “string”/>
   <element name=”Numero” type= “positiveInteger”/>
   <element name=”Bairro” type=”string”/>
  </sequence>
  <attribute name=”Estado” type=”UF”
</complexType>
```

Figura 1-38. Declaração de um tipo de dado complexo utilizando o indicador de ordem *sequence.*

O elemento *sequence* apresentado Figura 1-38 estabelece a ordem na qual os elementos do tipo de dado complexo devem aparecer. Além deste elemento, a linguagem disponibiliza outros dois indicadores de ordem: *all* e *choice.* O indicador *all* estabelece que os elementos definidos dentro do elemento *all* podem aparecer em qualquer ordem, mas cada um deve aparecer somente uma única vez (Figura 1-39). Já o indicador *choice* define que deve ser escolhido um entre os elementos que se encontram definidos dentro do elemento *choice* (Figura 1-40).

```
<complexType name=”Endereço”>
   <all>
     <element name=”Rua” type= “string”/>
     <element name=”Numero” type= “positiveInteger”/>
     <element name=”Bairro” type=”string”/>
   </all>
   <attribute name=“Estado” type= “UF”
</complexType>
```

Figura 1-39. Declaração de um tipo de dado complexo utilizando o indicador de ordem *all*.

```
<complexType name=“Pessoa”>
 <choice>
  <element name=“Aluno” type= “string”/>
  <element name=“Professor” type= “string”/>
 </choice>
</complexType>
```

Figura 1-40. Declaração de um tipo de dado complexo utilizando o indicador de ordem *choice*.

Na definição de um esquema, caso seja necessário, são definidos novos tipos de dados e estes são utilizados pelos elementos que vão compor o esquema. Além de permitir a declaração de novos tipos de dados simples e complexos concretos, a linguagem XML *schema* permite que sejam criados tipos de dados complexos abstratos, ou seja, tipos de dados que só podem ser utilizados por outros tipos de dados para derivação. Para determinar que um tipo de dado seja abstrato, o atributo *abstract* do elemento *complexType* deve estar com o valor *true*.

```
<complexType name=“Pessoa” type=“string” abstract=“true”>
   <sequence>
     <element name=“Nome” type=“string”/>
     <element name= “End” ref=“Endereço”/>
   </sequence>
</complexType>
```

Figura 1-41. Declaração de um tipo de dados complexo abstrato.

1.6. Derivação de Tipos

Da mesma forma que a linguagem disponibiliza meios de especificar que um tipo de dado só pode ser usado por derivação, ela permite proibição da derivação de um tipo de dado. Esta proibição pode ser feita para qualquer tipo de dado especificado, seja ele simples ou complexo. Para tal, é necessário que o atributo *final* do tipo de dado especificado tenha o valor *all*, *extension* ou *restriction*, caso seja um tipo de dado complexo, e *all*, *restriction*, *union* ou *list*, se for um tipo simples. *All* significa que o tipo de dado não pode ser derivado de nenhuma forma, *extension* que o tipo de dado não pode ser derivado por extensão, *restriction* que o tipo não pode ser derivado por restrição, *union* que o tipo de dado não pode ser utilizado na especificação de um novo tipo de dado simples que seja uma união de outros tipos, e *list* que o tipo de dado não pode ser utilizado para a especificação de um novo tipo de dado simples que seja uma lista de outros tipos. Por *default*, o valor do atributo *final* dos tipos de dados e elementos do esquema, é vazio. Porém, é possível que um novo valor *default* seja especificado para todos os elementos e tipos de dados do esquema. Para isso, deve-se estabelecer um valor ao atributo *finalDefault* do elemento *schema*. Os possíveis valores deste atributo são *restriction* ou *extension*. Para tipos simples de listas ou uniões, quando o atributo *finalDefaul* for definido com o valor *extension*, terá o mesmo efeito de estabelecer o atributo *final* destes com o valor *list* ou *union* respectivamente.

A Figura 1-42 declara um tipo complexo que proíbe a derivação dele por outro tipo de dado, através de restrição.

```
<complexType name="Address" final="restriction">
  <sequence>
    <element name="name"   type="string"/>
    <element name="street" type="string"/>
    <element name="city"   type="string"/>
  </sequence>
</complexType>
```

Figura 1-42. Proibição de derivação por restrição

A linguagem XML *schema* permite criar novos tipos de dados complexos a partir da derivação por extensão ou restrição de outros tipos de dados simples ou complexos. Derivação por extensão consiste em especificar um novo tipo de dado

idêntico a de outro tipo, mas com características adicionais. Já a derivação por restrição é quando se deseja especificar um novo tipo de dado que é idêntico a de outro tipo, mas com restrições impostas sobre os mesmos, ou seja, este novo tipo aceita um subconjunto do tipo que ele é derivado. Quando se especifica um novo tipo de dado que tenha seu conteúdo do tipo simples, mas que contém atributos (por isso é chamado de complexo), utiliza-se o elemento *simpleContent*. Este novo tipo de dado complexo é então derivado por extensão (Figura 1-43) ou restrição (Figura 1-44) de outro tipo de dado simples.

```
<element name="internationalPrice">
  <complexType>
    <simpleContent>
      <extension base="decimal">
        <attribute name="currency" type="string"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
```

Figura 1-43. Derivação por extensão

```
<element name="intRestrito">
  <complexType>
    <simpleContent>
      <restriction base="xs:integer">
        <minInclusive value="0"/>
        <maxExclusive value="100"/>
      </restriction>
    </simpleContent>
  </complexType>
</element>
```

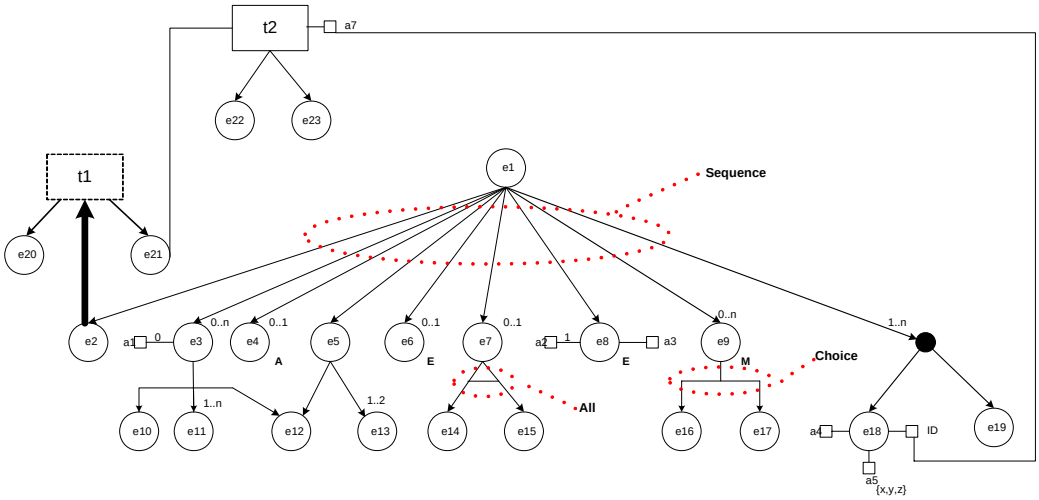
Figura 1-44. Derivação por restrição

Já quando o novo tipo de dado é derivado de outro tipo de dado complexo, utiliza-se o elemento *complexContent*.

1.7. Representação Gráfica de XML Schema

Com o objetivo de facilitar o entendimento de um esquema para XML, é apresentado neste trabalho uma representação gráfica para os conceitos do modelo da XML Schema. Vale ressaltar que esta representação abstrai detalhes de implementação, como tipos de dados e restrições de integridade. Estes detalhes são definidos em uma representação física ou implementação de esquemas XML. Esta representação se preocupa em descrever apenas a organização estrutural de dados XML.

A representação utilizada é uma extensão do modelo apresentado em [MEL05], tendo sido incluídas representações associadas aos conceitos específicos de XML Schema, visto que o modelo original tem como base DTD. Um exemplo desta representação é mostrado na Figura 3-4 (a).



(a)

```
<?xml version="1.0" encoding="UTF-8"?>
<e1>
  <e2>
    <e20></e20>
    <e21>1</e21>
  </e2>
  <e3>
    <e12>11.1</e12>
  </e3>
  <e5>
    <e12>11.1</e12>
    <e13>08:00:30</e13>
  </e5>
  <e8 a2="atributo2"/>
  <e18 a4="2005-10-12" a5="y" a6="valueJ"/>
  <e19>valueK</e19>
</e1>
```

(b)

Figura 1-45. Uma representação gráfica do esquema (a) e uma instância XML (b).

A representação baseia-se numa organização hierárquica em árvore utilizando elementos gráficos para representar os conceitos do modelo XML, que são os seguintes:

- **Elementos:** cada elemento presente em um esquema XML é representado por um círculo contendo o nome do elemento;
- **Relacionamento hierárquico entre os elementos:** os elementos complexos que possuem sub-elementos são interligados através de linhas direcionadas do elemento para seus sub-elementos, como mostra a Figura 1-45 (a);
- **Relacionamento de derivação por extensão:** o relacionamento entre os elementos ou tipos abstratos com outros elementos que o estendem é representado graficamente por uma seta dupla direcionada do elemento que está estendendo para o que está sendo estendido, como mostra a Figura 1-45;
- **Indicadores de Ordem:** para representar os indicadores de ordem *sequence*, *all* e *choice*, usados para construir elementos complexos, são definidas linhas de conexão diferentes entre os elementos, conforme pode ser observado na Figura 1-45(a). O conceito de ordem dos elementos definido pelo indicador *sequence* é estabelecido pela representação gráfica da esquerda para a direita, levando em consideração a posição dos elementos pertencentes ao *sequence*;
- **Elementos *mixed, empty e anyType:*** Os tipos de elementos *mixed*, *empty* e *anyType* são indicados respectivamente pelas letras M, E e A colocadas ao lado do elemento;
- **Atributos**: estes são representados por um quadrado ligado ao elemento ao qual pertence e o seu nome é identificado ao lado. Para representar as opções presentes em um tipo de dado definido pela faceta *enumeration,* coloca-se as opções entre chaves abaixo do nome do atributo, como no atributo a5 do elemento e18 na Figura 1-45(a);
- **Indicadores de ocorrência**: a ocorrência de elementos e atributos em instâncias XML pode ser definida com o uso de indicadores de ocorrências. Na XML Schema, por exemplo, eles são indicados por *maxOccurs* e

minOccurs para elementos, e pelo atributo *use* para atributos. Na representação lógica, a ocorrência para elementos é indicada pela definição do limite inferior e superior. Atributos são por definição opcionais, podendo ser especificados também como obrigatórios ou proibidos. Para representar estes conceitos são utilizados “0” para proibido e “1” para obrigatório sobre a linha que conecta o atributo ao elemento e a ausência desta indicação sugere que o atributo é opcional;

- **Grupos**: grupos de elementos aninhados dentro de outro elemento podem ser definidos. Para representá-los é utilizado um ponto. Os elementos pertencentes ao grupo são ligados ao ponto utilizando um dos modelos de linhas de conexões definidos para os indicadores de ordem (*all*, *choice* e *sequence*) visto que grupos sempre utilizam internamente um destes indicadores;
- **Tipos abstratos**: são representados por um retângulo pontilhado contendo o nome do tipo como apresentado pelo tipo t1 definido na Figura 1-45 (a).
- **Tipos complexos**: são representados da mesma forma que os tipos abstratos, só que com retângulos contínuos. O tipo T2 da Figura 1-45(a) é um exemplo deste conceito. A ligação entre os tipos complexos e os elementos que são do tipo é fieta através de uma linha simples.
- **Atributos tipo ID e IDREF**: estes atributos possuem uma ligação entre eles que deve ser representada. Esta representação gráfica representa esta ligação através de uma linha simples que os interliga. Um exemplo da representação utilizada para ligação destes atributos pode ser visualizada no atributo a7 do tipo IDREF de T2 da Figura 1-45, que está ligado com o atributo ID do elemento e18.

A Figura 1-46 mostra um esquema XML definido em XML Schema que corresponde ao esquema lógico da Figura 1-45 (a), conforme a representação gráfica adotada. A Figura 1-45 (b) apresenta uma instância XML que está de acordo com estes esquemas.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
 <xs:element name="e12" type="xs:float"/>
  <xs:complexType name="t1" abstract="true">
    <xs:sequence>
      <xs:element name="e20" type="xs:string"/>
      <xs:element name="e21" type="xs:t2"/>
    </xs:sequence>
```

```
 </xs:complexType>
 <xs:complexType name="t2">
   <xs:sequence>
     <xs:element name="e22" type="xs:string"/>
     <xs:element name="e23" type="xs:int"/>
   </xs:sequence>
   <xs:attribute name="a7" type="xs:IDREF"/>
 </xs:complexType>
<xs:group name="g1">
 <xs:sequence>
  <xs:element name="e18">
   <xs:complexType>
      <xs:attribute name="a4" type="xs:date"/>
      <xs:attribute name="a5">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="x"/>
        <xs:enumeration value="y"/>
        <xs:enumeration value="z"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="ID" type="xs:ID"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="e19" type="xs:string"/>
  </xs:sequence>
</xs:group>
<xs:element name="e1">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="e2">
    <xs:complexType>
     <xs:complexContent>
      <xs:extension base="t1"/>
     </xs:complexContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="e3" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:choice>
      <xs:element name="e10" type="xs:integer"/>
      <xs:element name="e11" type="xs:string" maxOccurs="unbounded"/>
      <xs:element ref="e12"/>
     </xs:choice>
     <xs:attribute name="a1" type="xs:string" use="prohibited"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="e4"  minOccurs="0"/>
   <xs:element name="e5">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="e12"/>
      <xs:element name="e13" type="xs:time" maxOccurs="2"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="e6" minOccurs="0">
    <xs:complexType/>     </xs:element>
   <xs:element name="e7" minOccurs="0">
    <xs:complexType>
     <xs:all>
      <xs:element name="e14" type="xs:string"/>
      <xs:element name="e15" type="xs:integer"/>
     </xs:all>
    </xs:complexType>
   </xs:element>
   <xs:element name="e8">
```

```
     <xs:complexType>
      <xs:attribute name="a2" type="xs:string" use="required"/>
      <xs:attribute name="a3" type="xs:string" use="optional"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="e9" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType mixed="true">
      <xs:choice>
       <xs:element name="e16" type="xs:string"/>
       <xs:element name="e17" type="xs:string"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:group ref="g1" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Figura 1-46. Um exemplo de esquema em XML Schema.

2. BIBLIOGRAFIA

[BRA04] BRADLEY, N.. The XML Companion. Harlow: Addison-Wesley Ltd., 2004.

[MAR01] MARTIN, Didier; et al. Professional XML. Rio de Janeiro: Editora Ciência Moderna Ltda, 2001. 1º. edição.

[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.

[W3C04] W3C Recomendation. *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

[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/