UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

EDWALDO RAMOS DE BRITO MONTEIRO

UMA FERRAMENTA PARA CARGA DE BANCOS DE DADOS RELACIONAIS A PARTIR DE FONTES DE DADOS XML

FLORIANÓPOLIS/SC
2011

EDWALDO RAMOS DE BRITO MONTEIRO

UMA FERRAMENTA PARA CARGA DE BANCOS DE DADOS RELACIONAIS A PARTIR DE FONTES DE DADOS XML

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciências da Computação.

Orientador: Profa. Dra. Patrícia Vilain
Co-orientador: Alexandre Jonatan B. Martins

FLORIANÓPOLIS/SC
2011

EDWALDO RAMOS DE BRITO MONTEIRO

UMA FERRAMENTA PARA CARGA DE BANCOS DE DADOS RELACIONAIS A PARTIR DE FONTES DE DADOS XML

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciências da Computação do Curso de Ciências da Computação da Universidade Federal de Santa Catarina e aprovado, em sua forma final, em Julho de 2011.

_____
Prof. Dr. Vitório Bruno Mazzola
Coordenador do Curso

Apresentada à Banca Examinadora, composta pelos Professores:

_____
Orientador: Profa. Dra. Patrícia Vilain
Universidade Federal de Santa Catarina

_____
Prof. Dr. Ronaldo dos Santos Mello
Universidade Federal de Santa Catarina

_____
Prof. Dr. Vitório Bruno Mazzola
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer aos meus pais, Adriano de Brito Monteiro e Iolanda Vitorina Ramos Monteiro, e irmã, Elba Regina Ramos Monteiro, pelo imenso apoio durante toda minha vida acadêmica e investimento para que eu pudesse ter uma boa e sólida formação. Sem eles, nada disto seria possível!

Agradeço também à professora e minha orientadora, Patrícia Vilain e ao meu co-orientador Alexandre Jonatan B. Martins, pela paciência e prestabilidade demonstradas, pelas orientações, revisões e correções.

Aos professores membros da banca avaliadora, Ronaldo dos Santos Mello, e Vitório Bruno Mazzola, por terem aceitado meu convite como membros da banca e auxiliado para a qualidade deste trabalho.

Por fim, a todos os meus amigos, colegas e companheiros de curso, por me ajudarem nesta caminhada durante todos os anos de minha graduação.

“Para conhecermos os amigos é necessário passar pelo sucesso e pela desgraça. No sucesso, verificamos a quantidade e, na desgraça, a qualidade”
(CONFÚCIO)

RESUMO

Com a evolução da internet, dos sistemas de informação e do comércio eletrônico, a troca de informações entre diferentes aplicações e sistemas se tornou um requisito cada vez mais frequente. Graças à sua grande flexibilidade e portabilidade, o XML nos últimos anos vem sendo cada vez mais aceito como um padrão para representar, transferir e manipular dados em diversas aplicações. Hoje em dia, o XML desempenha um papel fundamental na integração de sistemas heterogêneos como, por exemplo, na troca de informações entre sistemas de banco de dados diferentes. Os sistemas de banco de dados relacionais são os mais utilizados na atualidade, sendo que grande parte da informação de negócio das empresas e organizações, está armazenada de acordo com o modelo relacional de dados.

Este trabalho consiste no desenvolvimento de uma ferramenta que possibilita a carga de bancos de dados relacionais diversos a partir de dados contidos em documentos XML. A ferramenta foi implementada usando a linguagem JAVA e suas principais tecnologias, tornando-a multi-plataforma e independente de SGBD.

Palavras-chaves: Bancos de dados relacionais, XML, mapeamento XML-Relacional, carga de BD relacionais

ABSTRACT

With the evolution of the Internet, information systems and e-commerce, the exchange of information between different applications and systems has become an increasingly frequent requirement. Thanks to its great flexibility and portability, XML has been widely accepted in the past years as the standard for representing, transferring and manipulating data in many applications. Nowadays, XML plays a fundamental role in integrating heterogeneous systems such as in the exchange of information between two different database systems. Relational database systems are the most common these days, and a lot of the business information is stored in organizations according to the relational model of data.

This work consists of developing a tool for performing the load of relational databases from data contained in XML documents. The tool was implemented using the JAVA language and its core technologies, which make it multi-platform and DBMS independent.

Keywords: Relational databases, XML, XML-relational mapping, load of relational databases

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

4GL – *4th Generation Language*
BD – *Banco de Dados*
BLOB - *Binary Large Object*
DOM - *Document Object Model*
CLOB – *Character Large Object*
DTD - *Document Type Definition*
EAI - *Enterprise Application Integration*
EDI - *Electronic Data Interchange*
EII - *Enterprise Information Integration*
ETL - *Extract, Transform, Load*
IDE – *Integrated Development Environment*
HTML - *HyperText Markup Language*
JAXP - *Java API for XML Processing*
JDBC - *Java Database Connectivity*
LDAP - *Lightweight Directory Access Protocol*
PCDATA - *Parsed Character Data*
ODBC - *Open Data Base Connectivity*
OOXML - *Office Open XML*
SAX - *Simple API for* XML
SGBD – *Sistema de Gerenciamento de Banco de Dados*
SGML - *Standard Generalized Markup Language*
SOA - *Service-Oriented Architecture*
SOAP - *Simple Object Access Protocol*
SQL - *Structured Query Language*
XBRL - *eXtensible Business Reporting Language*
XHTML - *eXtensible Hypertext Markup Language*
XML - *Extensible Markup Language*
XPath - *XML Path Language*
XSD - *XML Schema* Definition
XSL:FO - *XSL Formatting Objects*
XSL - *eXtensible Stylesheet Language*
XSLT - *eXtensible Stylesheet Language for Transformation*
W3C - *World Wide Web Consortium*
WSDL - *Web Services Description Language*

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho visa desenvolver uma ferramenta multi-plataforma e independente de SGBD, que possibilite fazer a carga de bancos de dados relacionais a partir de dados contidos em documentos XML.

Este capítulo apresenta uma introdução do que é elaborado neste trabalho, introduzindo ao leitor o contexto do mesmo, a motivação e justificativa para a sua elaboração, bem como os objetivos a serem alcançados.

1.1 Contextualização do tema

O XML (*Extensible Markup Language*) desempenha um papel cada vez mais importante como sendo um formato genérico de troca de dados, sobressaindo-se sobre outros formatos (W3C; 2011). O seu suporte ao UNICODE torna-o altamente portável e capaz de representar praticamente qualquer *string*. Além disto, o uso de *tags* para rotular dados faz com que documentos XML sejam auto-descritivos, podendo ser facilmente lidos por humanos e ao mesmo tempo usados por diferentes aplicações. O formato padronizado de um documento XML possibilita que o mesmo possa ser lido por qualquer aplicação.

Com a evolução da internet, dos sistemas de informação e do comércio eletrônico, a troca de informações entre diferentes aplicações se tornou um requisito cada vez mais frequente, sendo necessário prover interoperabilidade entre sistemas heterogêneos.

Ainda de acordo com a W3C (2011), o principal uso do XML é de transferência de dados entre aplicações distintas e em ambientes heterogêneos. Antes da “era XML”, as empresas que necessitavam de transferência de dados, utilizavam o EDI (*Electronic Data Interchange*), onde um grande número de transações era comunicado em arquivos em lote. Obviamente as empresas tinham de negociar previamente padrões pouco flexíveis para os arquivos em lote. Hoje em dia, com a flexibilidade e portabilidade do XML, uma aplicação pode gerar documentos XML a partir de dados armazenados em um banco de dados (BD), e estes documentos podem facilmente ser lidos posteriormente por outra aplicação, e os dados contidos nos mesmos podem ser

extraídos e armazenados em um modelo de dados possivelmente diferente do BD inicial.

Os BDs relacionais são os bancos de dados mais utilizados na atualidade, sendo que grande parte da informação de negócio das empresas e organizações está armazenada de acordo com o modelo relacional de dados. A troca de dados entre documentos XML e bancos relacionais é algo possível e muito importante na integração de sistemas de informação. Neste sentido, existem softwares desenvolvidos especificamente para realizar a carga de BD relacionais a partir de dados contidos em documentos XML, e também para realizar a extração de dados armazenados no BD e armazená-los em documentos XML.

1.2 Motivação e Justificativa

Graças à sua grande flexibilidade e portabilidade, o XML nos últimos anos vem sendo cada vez mais aceito como um padrão para representar, transferir e manipular dados em diversas aplicações. Representação de dados em aplicações de gerenciamento de conteúdo, aplicações de transações bancárias e de comércio eletrônico, são alguns exemplos de sua aplicação.

Hoje em dia o XML desempenha um papel fundamental na integração de sistemas heterogêneos como por exemplo, na troca de informações entre dois sistemas de BD diferentes, usando documentos XML para o intercâmbio dos dados. Como os sistemas de BDs relacionais são os mais usados nos dias de hoje, torna-se necessário fornecer mecanismos para que seja possível fazer a carga dos mesmos a partir de fontes de dados XML**.**

Alguns dos principais Sistemas de Gerenciamento de Banco de Dados (SGBD) relacionais existentes no mercado atualmente disponibilizam extensões para extrair dados de documentos XML e armazená-los no BD e vice versa. Porém, na sua grande maioria, são soluções comerciais pagas, de código fechado, limitadas e pouco flexíveis. Geralmente exigem um formato específico ou proprietário para os documentos XML e obviamente a transferência só pode ser feita através do SGBD em questão (BEZA et al; 2008).

Algumas das ferramentas mais poderosas disponíveis, que permitem armazenar dados contidos num documento XML em um BD relacional também são soluções

proprietárias, de código fechado e licença comercial e a maioria se restringe a um conjunto limitado de SGBD.

Das soluções gratuitas disponíveis, a maioria é complexa e possui várias funcionalidades além da carga de BDs relacionais a partir de documentos XML, o que se torna um inconveniente para o usuário quando o mesmo quer uma solução simples e de uso fácil, que o permita armazenar dados em BDs relacionais a partir de documentos XML.

Neste contexto, surge a seguinte questão: Como oferecer uma solução que seja simples e ao mesmo tempo multi-plataforma e independente de SGBD?

Este trabalho visa descrever as soluções já existentes e apresentar uma nova ferramenta de uso fácil, independente de plataforma e de código aberto, em uma tentativa de propor uma alternativa às ferramentas já existentes. A ideia é oferecer ao usuário uma solução multi-plataforma, simples de usar e com apenas o necessário para fazer a carga de qualquer BD relacional a partir de documentos XML.

1.3 Objetivos

O objetivo geral deste trabalho consiste no desenvolvimento de uma ferramenta para carga de BDs relacionais diversos a partir de dados contidos em documentos XML. Para tanto, os seguintes objetivos específicos foram estabelecidos:

a) Fazer um estudo dos trabalhos relacionados e ferramentas já existentes;
b) Especificar uma linguagem de mapeamento XML-Relacional;
c) Projetar e desenvolver a ferramenta proposta;
d) Validar a ferramenta proposta.

1.4 Metodologia

Uma pesquisa pode ser classificada quanto à natureza, a forma de abordagem, aos objetivos e aos procedimentos técnicos.

Quanto à natureza, o presente trabalho se classifica como uma pesquisa aplicada, já que de acordo com Silva (2004), a pesquisa aplicada tem por objetivo gerar conhecimentos para aplicação prática dirigida à solução de problemas específicos.

Quanto à forma de abordagem, trata-se de uma pesquisa qualitativa. Em Silva (2004, p. 14), a pesquisa qualitativa é descrita como

> Pesquisa Qualitativa: considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números. A interpretação dos fenômenos e a atribuição de significados são básicos no processo de pesquisa qualitativa. Não requer o uso de métodos e técnicas estatísticas. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento chave. É descritiva. Os pesquisadores tendem a analisar seus dados indutivamente. O processo e seu significado são os focos principais de abordagem.

Quanto aos objetivos, pode-se classificar como pesquisa exploratória, que "visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses. [...] Assume, em geral, as formas de Pesquisas Bibliográficas e Estudos de caso." (SILVA, 2004).

Quanto aos procedimentos técnicos, trata-se de um trabalho de estudo de caso e pesquisa bibliográfica. De acordo com Silva (2004), estudo de caso é "quando envolve o estudo profundo e exaustivo de um ou poucos objetos de maneira que se permita o seu amplo e detalhado conhecimento" e pesquisa bibliográfica "quando é elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e atualmente com material disponibilizado na Internet".

Na etapa inicial do projeto foi realizada a revisão bibliográfica. A primeira parte da pesquisa foi embasada em artigos, livros e documentos publicados na internet que abordam BD relacionais, SGBD, XML e suas tecnologias, integração de sistemas, etc . Em seguida foi feito um estudo dos trabalhos relacionados e ferramentas já desenvolvidas e disponíveis no mercado.

A próxima etapa consistiu em estudar as tecnologias utilizadas no desenvolvimento da ferramenta, os seus usos e suas aplicações.

Por fim, deu-se início à definição da linguagem de mapeamento, ao projeto da ferramenta, e consequentemente à implementação da mesma.

2 CONCEITOS BÁSICOS

Neste capítulo, são descritos alguns conceitos que dão o embasamento teórico necessário para o entendimento do trabalho desenvolvido.

Vale ressaltar que os conceitos e tecnologias aqui apresentados são complexos, sendo que a descrição profunda e detalhada dos mesmos foge ao escopo deste trabalho, sendo apresentado apenas o que se considera necessário para a correta compreensão do restante do trabalho.

2.1 XML

De acordo com a W3C (*World Wide Web Consortium*), o XML (eXtensible Markup Language) é uma linguagem de marcação simples e muito flexível derivada da SGML (*Standard Generalized Markup Language*). Trata-se de uma recomendação da própria W3C para a criação de documentos com dados organizados hierarquicamente, tais como textos, BD ou desenhos vetoriais. A linguagem é classificada como extensível porque permite definir os elementos de marcação.

Uma linguagem de marcação é um agregado de códigos que são aplicados a dados ou textos para serem lidos por computadores ou pessoas. O HTML (*HyperText Markup Language*) também é uma linguagem de marcação, porém o mesmo pode ser visto como uma aplicação da SGML enquanto que o XML é uma simplificação dela. O XML é um subconjunto da SGML e é capaz de descrever diversos tipos de dados. Um dos seus principais objetivos é o de facilitar o compartilhamento de dados através da internet e a integração entre diferentes sistemas de informação. Trata-se de uma linguagem simples, porém poderosa, que combina a flexibilidade da SGML com a simplicidade da HTML.

As principais características da linguagem XML são:

a) Possibilidade de criar um número ilimitado de tags;
b) Foco na estrutura da informação, e não na sua aparência;
c) É simples e legível tanto para computadores como por seres humanos;
d) Facilita a interligação entre diferentes BDs;
e) Permite a criação de arquivos para validar a estrutura dos dados.

Pela facilidade que o XML trouxe ao compartilhamento de informações entre aplicações diferentes, e devido ao surgimento de várias tecnologias relacionadas que o complementam, a linguagem rapidamente se popularizou e hoje é um padrão de fato e largamente utilizado no armazenamento e transferência de dados entre diferentes aplicações e sistemas heterogêneos.

Um documento XML é um texto em formato Unicode com tags de marcação e outras informações. Para Furtado (2003),

> Um documento XML é uma árvore rotulada onde um nó externo consiste de:
>
> a) Dados de caracteres (uma sequência de texto);
> b) Instruções de processamento (anotações para os processadores), tipicamente no cabeçalho do documento;
> c) Um comentário (nunca com semântica acompanhando);
> d) Uma declaração de entidade (simples macros);
> e) Nós DTD (*Document Type Declaration*).
>
> E um nó interno é um elemento, o qual é rotulado com:
>
> a) Um nome;
> b) Um conjunto de atributos, cada qual consistindo de um nome e um valor.

Um documento XML começa com uma declaração XML, que é uma instrução de processamento. As instruções de processamento passam informações aos processadores de XML. As suas anotações são delimitadas por <? e ?>. Existe uma instrução de processamento especial, que começa por "xml". Um documento XML pode conter elementos simples e complexos. Um elemento é simples quando contém apenas texto, e é complexo (ou composto) quando contém um ou vários elementos e atributos (HAROLD et al; 2004).

Os dados estão contidos num único elemento, chamado de elemento raiz. Um elemento pode ter duas formas:

<elemento lista_atributos> conteúdo </elemento>

Ou

<elemento lista_atributos />

As anotações são sensíveis a maiúsculas e minúsculas e são sempre delimitadas por parêntesis angulares (< >). Existem algumas regras para os nomes dos elementos e de atributos. São elas:

a) Devem começar por uma letra ou pelo traço inferior.

b) Podem conter letras, dígitos e traços inferiores.
c) Espaços não são permitidos.

A lista de atributos de um elemento é opcional. Existindo, ela pode conter um ou mais atributos. Cada atributo possui um nome e um valor. A notação é *nome_do_atributo=”valor”.* No exemplo a seguir, o elemento pessoa, tem dois atributos: nome e idade:

<pesssoa nome=“Maria” idade=28> ... </pessoa>

Existem diferentes tipos de elementos, classificados quanto ao conteúdo dos mesmos:

a) Elementos vazios: não têm nenhum elemento filho nem qualquer elemento textual.
b) Elementos textuais: contêm apenas texto.
c) Elementos estruturados: contêm apenas outros elementos.
d) Elementos mistos: contêm texto e outros elementos.

Em um documento XML são considerados comentários todos os blocos de texto delimitados por <!-- e --> e não podem aparecer antes da declaração XML do início. Além disso não podem aparecer dentro de uma anotação e nem é permitido usar dois hífens seguidos (--) dentro de um comentário.

Exemplo de documento XML:

```
<?xml version="1.0" ?>
<!-- Isto é um comentário -->
<mensagem>
        <para>Maria</para>
        <de>Jorge</de>
         <titulo>Lembrete</titulo>
         <corpo>Hoje é o dia da festa de aniversário da Aline</corpo>
</mensagem>
```

De acordo com HAROLD et al (2004, pag 53), para um documento XML ser considerado bem formado, é preciso que o mesmo obedeça às seguintes regras:

a) Tem de incluir uma declaração XML no seu início;

b) Tem de incluir um ou mais elementos, e o primeiro, elemento raiz, tem de incluir todos os outros;
c) Todos os elementos têm marcas de início e de fim a menos que o elemento seja vazio, e neste caso a marca de início termina por ‘/>’;
d) Todos os elementos devem estar aninhados de forma correta;
e) Os valores dos atributos têm de ser delimitados por aspas ou apóstrofos.
f) Tem de ter um único elemento raiz;
g) As anotações iniciais e finais precisam combinar. Lembrando que o XML é sensível a maiúsculas e minúsculas;
h) Não pode conter atributos repetidos;
i) Possui apenas identificadores válidos para elementos e atributos.

2.1.1 Tecnologias relacionadas

O XML é uma tecnologia relativamente simples que possui várias outras tecnologias complementares que, juntas, fazem do XML um padrão tão poderoso. A seguir são descritas de forma breve algumas das tecnologias relacionadas ao XML.

2.1.1.1 Tecnologias de validação

Uma das grandes vantagens do XML é a possibilidade de criar tags próprias, o que traz grande liberdade na criação de documentos XML. Essa liberdade, no entanto, pode trazer grandes problemas se não for devidamente controlada. Documentos XML contendo a mesma informação, podem ter estruturas muito diferentes. Muitas vezes é inviável permitir toda essa flexibilidade do XML na transmissão de dados sem fazer o controle de qual estrutura de um documento deve ser válida. Deve haver um acordo prévio sobre o padrão a ser utilizado na estrutura dos documentos. Neste sentido, surgiram algumas tecnologias que solucionam este problema e permitem verificar se um documento XML é valido, de acordo com regras definidas previamente.
Um documento XML bem formado não necessariamente é válido para uma determinada aplicação.

Um esquema XML é um modelo que contém a descrição de todos os elementos, tipos de dados, atributos, entidades e suas relações. Diz-se que um documento XML é

válido em relação a um determinado esquema quando obedece a todas as regras descritas no esquema. A existência de um esquema permite a validação de um documento XML fazendo uso de algumas tecnologias complementares ao XML.

As duas tecnologias mais usadas e conhecidas para a validação de documentos XML são o DTD e o XSD.

O **DTD** (*Document Type Definition*) especifica um conjunto de regras que define a estrutura de um documento. Trata-se de um conjunto de declarações de marcação que define um tipo de documento para linguagens de marcação da família SGML, o que inclui o XML e o HTML. (VIGNATTI; 2009)

O DTD usa uma sintaxe formal e que especifica quais elementos e atributos podem estar presentes em um documento XML, em que parte do documento e qual o possível conteúdo dos mesmos.

Um DTD pode ser declarado dentro de um documento XML (no prólogo) ou como referência externa, em um documento separado.

Segundo a W3CSCHOOLS, as regras especificadas por um DTD e que definem um documento XML incluem:

a) Declaração de um conjunto de elementos. Não é possível utilizar elementos que não estejam inclusos neste conjunto.
b) Definição do conteúdo para cada elemento. São especificados quais elementos ou dados que determinado elemento XML pode conter, em qual ordem, quantidade e se é opcional ou obrigatório.
c) Declaração de um conjunto de atributos para cada elemento. A declaração define o nome, o tipo, valores padrões, se for o caso, e comportamento de cada atributo (obrigatório ou opcional).

O exemplo a seguir exemplifica a sintaxe do DTD. No exemplo é declarado um elemento complexo de nome ‘mensagem’:

```
<!ELEMENT mensagem (para, de, assunto, corpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT assunto (#PCDATA)>
<!ELEMENT corpo (#PCDATA)>
```

Durante muito tempo o DTD foi o padrão de validação de documentos XML mais utilizado, porém algumas de suas limitações fizeram com que se tornasse

necessário criar novos padrões que fornecessem maior poder na validação e restrição dos dados. O DTD não oferece um conjunto amplo de tipos. Todos os dados são interpretados como texto puro. Além disso, o DTD também não suporta espaços de nomes (*namespace*), e força que os elementos sempre apareçam na ordem especificada (W3CSCHOOLS).

Neste sentido, o **XSD** (XML Schema Definition), ou apenas XML Schema, se tornou o padrão e recomendação da W3C para validação de documentos XML, em 2001. Além de conseguir suprir as limitações do DTD citadas anteriormente, este novo padrão fornece diversas novas funcionalidades.

O XML Schema descreve a estrutura de um documento XML e define um tipo de documentos. Permite definir:

a) Quais elementos podem aparecer em um documento XML;
b) Quais atributos podem aparecer em cada um dos elementos;
c) Quais os elementos descendentes de cada elemento, a ordem e o número;
d) Se os elementos podem ser vazios ou conter texto;
e) Os tipos de dados dos elementos e atributos;
f) Valores fixos para elementos e atributos e valores por omissão.

Entre as novas funcionalidades introduzidas por este padrão, se destaca a introdução de tipos de dados. O XSD dá suporte a tipos primitivos de dados e permite a expansão desses tipos, criando novos, também chamados tipos derivados. (W3CSCHOOLS).

Entre os tipos primitivos oferecidos pelo XSD os principais são:

a) string - uma string de qualquer tamanho;
b) integer - qualquer numero inteiro;
c) decimal - numero decimal de precisão arbitrária;
d) float - numero de ponto flutuante de precisão simples (32 bits);
e) double - numero de ponto flutuante de precisão dupla (64 bits);
f) date - uma data (ano-mês-dia);
g) time - uma hora (hora-minuto-segundo);
h) duration - duração num formato específico;
i) gDay - dia no calendário Gregoriano;
j) gMonth - mês no calendário Gregoriano;
k) gYear - ano no calendário Gregoriano;

Outro recurso interessante que o XSD acrescentou em relação ao DTD é a possibilidade de usar namespaces. Pelo fato do XML permitir ao usuário grande flexibilidade na definição de tags próprias, é bastante comum, que usuários diferentes escolham as mesmas tags, porém estas podem ter significados totalmente diferentes.

Para contornar este problema o XSD introduziu o conceito de namespaces. Um namespace é um *container* para os nomes utilizados. Ele tem a função de agrupar os nomes e geralmente tem seu nome associado a um nome único (geralmente uma URL).

De acordo com a W3CSCHOOLS, para definir o conteúdo de um elemento complexo o XSD oferece indicadores de ordem e de ocorrência. Os indicadores de ordem são os seguintes:

a) sequence – indica que o conteúdo do elemento é formado por outros elementos e especifica a ordem a qual estes devem obedecer;
b) choice – indica que o conteúdo do elemento é formado por um e apenas um dos elementos especificados;
c) all - indica que o elemento é constituído por outros elementos (filhos) que podem aparecer ou não (0 ou 1 vez) e em qualquer ordem.

A especificação do número de ocorrências de um determinado elemento é feita através dos indicadores *minOccurs* e *maxOccurs*, que especificam a quantidade mínima e máxima que um elemento pode aparecer em um documento, respectivamente.

A seguir é exemplificada a sintaxe do XSD. No exemplo, novamente é declarado um elemento complexo de nome mensagem. Note-se o uso do indicador sequence, que implica que os elementos de nome ‘para’, ‘de’, ‘assunto’ e ‘corpo’, precisam aparecer no documento na ordem especificada.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema    xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="mensagem">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="para" type="xs:string"/>
                         <xs:element name="de" type="xs:string"/>
                         <xs:element name="assunto" type="xs:string"/>
                        <xs:element name="corpo" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

2.1.1.2 XSLT

XSLT (*eXtensible Stylesheet Language for Transformation*) é uma linguagem de marcação baseada em XML, usada para transformar um documento XML em outro documento XML, ou em algum outro tipo de documento qualquer, como por exemplo um HTML ou até mesmo um texto puro. Nesta transformação, o documento original não sofre alterações e um novo documento é criado com base no conteúdo do documento existente. (W3CSCHOOLS)

Na maioria das vezes, o XSLT é usado para converter dados entre diferentes esquemas XML e para converter dados XML em páginas web ou documentos PDF (*Portable Document Format*).

Assim como o XML e o HTML, a especificação XSLT também é uma recomendação desenvolvida pela W3C.

O princípio de funcionamento do XSLT é similar ao das folhas de estilos CSS (*Cascading Style Sheets)*, que apenas determinam a forma com que um documento HTML ao qual elas estão associadas será apresentado num navegador, sem modificá-lo. No entanto, existe uma diferença crucial entre o XSLT e o CSS. Um documento XSL pode acrescentar ou esconder conteúdo na apresentação do documento XML, de forma que a transformação é mais poderosa do que a com folhas de estilo CSS.

Assim sendo, ao transformar um documento XML em outro, o XSLT pode adicionar e/ou remover elementos e atributos, ordenar ou reestruturar elementos, realizar testes e tomar decisões sobre quais elementos esconder ou mostrar no documento de saída. (W3CSCHOOLS)

2.1.1.3 XPATH

XPath (*XML Path Language*) é uma linguagem de consulta que permite selecionar nós de um documento XML. Pode ainda ser usado para calcular valores, a partir do conteúdo de um documento XML. (W3CSCHOOLS)

Trata-se de uma linguagem para recuperar partes de um documento XML além de permitir que se navegue pelo mesmo, através de um sistema de caminhos inspirado nos nomes e caminhos dos sistemas de arquivos.

Assim como o XSLT, trata-se de uma recomendação do W3C e desde 2007 se encontra na sua versão 2.0. Tem diversas aplicações em outras especificações XML como o XSLT, por exemplo.

O padrão XPath apresenta o conceito de Nó. Qualquer documento XML é formado por nós. Existe o nó raiz, nós intermediários, nós internos a outros nós, etc. Um nó pode ser do tipo elemento, atributo, comentário, instrução de processamento, texto ou namespace.

A navegação em um documento XML pelo XPath se parece bastante com a navegação em uma estrutura de diretórios ou de links da internet. (W3CSCHOOLS)

2.1.2 XML e bancos de dados

De acordo com Bourret (2005), existem duas formas distintas de armazenar documentos XML em um BD. A primeira consiste em fazer o mapeamento do esquema do documento para o esquema do BD e assim fazer a carga do BD de acordo com este mapeamento. A segunda forma é usar um conjunto fixo de estruturas que pode armazenar qualquer documento completo, sem a necessidade de conversões e mapeamentos.

Sistemas de BDs que suportam o primeiro método, são designados de sistemas de bancos de dados com suporte a XML, ou, em inglês, *XML-enabled database.* Este tipo de SGBD faz o mapeamento de instâncias do modelo de dados do XML para instâncias do seu próprio modelo de dados, como, por exemplo, o modelo relacional. Documentos XML são aceitos como entrada e a carga do BD é feita a partir desses dados. Porém, para que este mapeamento seja possível, a estrutura do documento XML é predefinida, específica e nada flexível. Esse tipo de SGBD geralmente também tem a capacidade de gerar documentos XML com formato predefinido a partir dos dados que estão armazenados no BD. Note-se que é o próprio SGBD, ou alguma extensão do mesmo, que faz as conversões necessárias para a carga do BD a partir de documentos XML ou a geração de documentos XML a partir de dados armazenados no BD.

Já os sistemas de BDs que suportam o segundo método são chamados de sistemas de BDs XML nativos, ou, em inglês, *native XML database.* O modelo interno deste tipo de BD depende do XML e usa documentos XML como unidade fundamental

de armazenamento, porém não necessariamente estes documentos são armazenados em forma de arquivos de texto. Utilizam o modelo de dados do XML diretamente, sem a necessidade de conversões e mapeamentos.

Segundo Bourret (2005), sistemas de BDs com suporte a XML são úteis quando se pretende extrair dados existentes no BD em forma de documentos XML ou importar dados de documentos XML e armazená-los em um banco já existente. No entanto, não são uma boa forma de armazenar documentos XML completos, pelo fato de descartarem informações do documento, comentários, instruções de processamento, etc., armazenando apenas os dados e a hierarquia dos mesmos.

Por fim, Bourret (2005) salienta o fato de que no caso dos sistemas de BDs com suporte a XML, a troca de dados entre o BD e documentos XML só é possível quando o esquema do documento XML é conhecido e suportado pelo SGBD. Por outro lado, sistemas de BDs XML nativos são capazes de armazenar qualquer documento XML completo, independentemente do esquema, e ainda suportam linguagens de consulta similares ao SQL, para a manipulação dos documentos armazenados. Apesar da existência dessas linguagens similares ao SQL, o padrão para consulta de documentos XML é o XQuery, que atualmente é a recomendação da W3C.

3 TRABALHOS RELACIONADOS

Com a crescente popularização do XML como um meio eficiente de troca de dados entre aplicações e integração de sistemas, e com o surgimento de novas tecnologias e ferramentas poderosas complementares ao XML, muito se tem desenvolvido no sentido de facilitar a troca de dados entre documentos XML e diferentes BDs.

Neste capítulo são descritos alguns softwares que permitem que dados contidos num documento XML sejam armazenados em BDs relacionais, bem como as principais características e a licença de uso dos mesmos.

Segundo Bourret (2010), tais softwares podem ser divididos em quatro categorias diferentes. São elas:

a) Middlewares;
b) IDE (*Integrated Development Environment*) e editores;
c) BDs com suporte a XML;
d) Softwares de integração de dados.

No restante deste capítulo cada uma das categorias acima citadas é detalhada e são apresentados alguns exemplos.

3.1 Middlewares

Um middleware é um software de terceiros, que é usado por aplicações para fazer a carga de BDs a partir de documentos XML. A aplicação invoca o middleware, que por sua vez faz todo o trabalho de carga do BD. A maioria dos middlewares acessa os dados em BD relacionais através de tecnologias como o ODBC (Open Data Base Connectivity) e JDBC (Java Database Connectivity).

A seguir são apresentados alguns middlewares existentes, agrupados de acordo com a licença de uso dos mesmos.

3.1.1 Comerciais

1. **ADO**

O ADO é um middleware desenvolvido pela *Microsoft* que permite a troca de dados entre documentos XML e BD relacionais usando o ODBC.

Este middleware utiliza um conceito de objeto *Recordset*, que é um objeto que armazena um conjunto de registros e colunas de uma tabela do BD relacional.

O ADO pode persistir um objeto Recordset como um documento XML e pode converter um documento XML em um objeto Recordset. Para tal, deve-se especificar o mapeamento do documento XML para objetos Recordset.

2. **Connect XML-2-DB**

Connect XML-2-DBL é um middleware escrito em Java e desenvolvido pela empresa *Skyhawk Systems*, para troca de dados entre documentos XML e o SQL Server ou Oracle. Ele usa um mapeamento que é descrito por uma linguagem baseada em XML.

3. **DbToXml**

O DbToXml é um middleware desenvolvido pela empresa *SoftRUs* que, ao contrário da maioria dos produtos da categoria, gera código a partir de um esquema de BD, em vez de um esquema XML. O DbToXml pode gerar esquemas XML, esquemas BizTalk, DTDs, amostras de documentos XML e scripts SQL para inserir, atualizar e excluir dados no BD usando o ODBC.

4. **DB/XML Transform**

O DB/XML Transform é um middleware desenvolvido pela empresa *DataMirror* que posteriormente veio a ser adquirida pela IBM, que permite a troca de dados entre BD relacionais, documentos XML e arquivos texto, em qualquer combinação possível. O acesso ao BD é feito através do JDBC e é fornecida uma interface gráfica para auxílio no mapeamento XML-Relacional.

5. Extreme Translator

O Extreme Translator é um conjunto de utilitários desenvolvidos pela *Etasoft*, para troca de dados entre uma grande variedade de formatos, tais como documentos XML, EDI, arquivos de texto, BD relacional, etc. O acesso ao BD é feito através do ODBC. O mapeamento entre o XML e o BD é feito através de um utilitário com interface gráfica.

6. SQLXML

SQLXML é um utilitário desenvolvido pela *Microsoft* que possibilita a troca de dados entre documentos XML e o Microsoft SQL Server. O nome não deve ser confundido com SQL/XML, que é a extensão ISO XML para a SQL.

O SQLXML usa um mapeamento entre o esquema XML e o esquema do BD. Tal mapeamento é especificado através de anotações em um documento XML Schema.

7. Allora

O Allora é um middleware desenvolvido pela empresa *HiT Software* que pode ser invocado pelas aplicações para a troca de dados entre um documento XML e um BD relacional. A comunicação com o BD é feita através do JDBC ou ODBC. Está incluída uma ferramenta com interface gráfica para se fazer o mapeamento de DTDs e XML Schemas para o esquema do BD.

8. Altova MapForce

O Altova MapForce é um middleware desenvolvido pela empresa Altova, que permite aos usuários fazer a troca de dados entre documentos XML, BDs relacionais, arquivos texto e documentos EDI. Os usuários definem mapeamentos com uma ferramenta gráfica, que então gera código XSLT 1.0, XSLT 2.0, XQuery, Java, C# ou C++ para realmente transferir os dados. A comunicação com o BD é feita através do JDBC ou ODBC.

3.1.2 Freewares

1. **Oracle XML Developer's kit (XDK)**

O XDK da Oracle é um conjunto de ferramentas para trabalhar com o XML. Uma das ferramentas incluídas e que permite a troca de dados dentre documentos XML e BDs é o XML SQL Utility.

O XML SQL Utility é um conjunto de classes Java para a troca de dados entre um BD relacional e um documento XML.

2. **xmlToSql**

O xmlToSql é um utilitário de linha de comando para armazenar os dados de um documento XML em um BD relacional, desenvolvido por Jim Kent. A estrutura do documento é predefinida e deve corresponder à estrutura das tabelas no BD.

A saída do xmlToSql é um diretório de arquivos e instruções SQL que podem ser usadas para armazenar os dados em qualquer BD relacional.

3.1.3 Open Source

1. **Castor**

Castor é um middleware desenvolvido pela exolab.org que tem a capacidade de preencher objetos Java a partir de documentos XML e serializar objetos Java em forma de documentos XML. Também pode fazer a troca de dados entre objetos Java e BDs relacionais.

Castor pode mapear automaticamente os objetos que seguem o padrão do Java Beans. Para outros objetos, é utilizada uma linguagem de mapeamento baseada no XML. O mapeamento é do tipo objeto-relacional.

2. DBIx::XML::DataLoader

O DBIx::XML::DataLoader é um módulo PERL desenvolvido por Christopher Berning para a carga de BDs relacionais a partir de um documento XML. Ele usa um arquivo de mapeamento para especificar como os dados devem ser transferidos.

3. Osage

O Osage é um middleware desenvolvido pelo George Stewart, que possui recursos para serialização de objetos em forma de documentos XML e o processo inverso também. Pode ser usado para fazer a carga de BDs a partir documentos XML através dos objetos intermediários. Os usuários escrevem documentos que mapeiam as classes para o BD. Estes documentos também especificam as relações entre os objetos e como esses relacionamentos são mapeados para o BD. O acesso ao BD é feito através do JDBC.

4. dtd2sql e xml2sql

O dtd2sql é um módulo Python que gera um conjunto de comandos CREATE TABLE do SQL a partir de um DTD, usando um arquivo de mapeamento.

O xml2sql também é um módulo Python. Ele gera um conjunto de instruções SQL INSERT a partir de um documento XML, usando novamente um arquivo de mapeamento. O usuário deve executar as instruções SQL para criar as tabelas e inserir os dados no BD relacional.

5. XQuare Bridge

XQuare Bridge é um middleware desenvolvido pela empresa Odonata para a troca de dados entre documentos XML e BDs relacionais. Ele usa uma linguagem proprietária baseada em XML de mapeamento entre o esquema XML e o esquema do BD. O acesso ao BD é feito através do JDBC.

3.1.4 Quadro comparativo

O Quadro 1 faz um resumo dos middlewares descritos anteriormente e mostra algumas diferenças básicas entre os mesmos como a licença de uso, o tipo de BD que suportam e o sentido da transferência de dados que é suportado, podendo ser do BD para um documento XML ou vice-versa.

Produto	**Desenvolvedor**	**Licença de uso**	**Tipo BD**	**BD para XML**	**XML para BD**
ADO	Microsoft	Comercial	Relacional	Sim	Sim
Allora	HiT Software	Comercial	Relacional	Sim	Sim
Altova MapForce	Altova	Comercial	Relacional	Sim	Sim
Castor	exolab.org	Open Source	Relacional	Sim	Sim
Connect XML-2-DB	Skyhawk Systems	Comercial	Relacional	Não	Sim
DbToXml	SoftRUs	Comercial	Relacional	Sim	Sim
DBIx::XML::DataLoader	Christopher Berning	Open Source	Relacional	Não	Sim
DB/XML Transform	IBM	Comercial	Relacional	Sim	Sim
Extreme Translator	Etasoft	Comercial	Relacional	Sim	Sim
Oracle XML Developer's Kit (XDK)	Oracle	Freeware	Relacional	Sim	Sim
Osage	George Stewart, et al	Open Source	Relacional	Sim	Sim
dtd2sql e xml2sql	David Mertz	Open Source	Relacional	Sim	Sim
xmlToSql	Jim Kent	Freeware	Relacional	Sim	Sim
SQLXML	Microsoft	Comercial	Relacional	Sim	Sim

XQuare Bridge	Odonata	Open Source	Relacional, XML	Sim	Sim

Quadro 1 – Comparação entre alguns dos principais middlewares de transferência de dados entre documentos XML e BDs relacionais

3.2 IDE e editores

Os IDEs, ou, em português, Ambientes Integrados de Desenvolvimento, e editores XML são softwares projetados para auxiliar no desenvolvimento de aplicações XML e para editar documentos XML, respectivamente.

Alguns desses softwares incluem bibliotecas que permitem a troca de dados entre documentos XML e BDs.

A seguir são apresentados alguns IDE e editores existentes, agrupados de acordo com a licença de uso dos mesmos.

3.2.1 Comerciais

1. **Stylus Studio**

Stylus Studio é um poderoso IDE para XML, desenvolvido pela empresa *DataDirect Technologies,* que suporta acesso ao BD.

Trata-se de uma suíte composta por centenas de ferramentas XML e uma das várias funcionalidades da IDE é permitir a troca de dados entre documentos XML e BD relacionais.

2. **Unicenter SQL-Station**

O Unicenter SQL-Station é um ambiente de desenvolvimento integrado desenvolvido pela *Computer Associates International* para criar, editar, testar, depurar e gerenciar código SQL e do BD do lado do servidor. Ele inclui um assistente para troca de dados entre documentos XML e BD, usando o XML SQL Utility da Oracle. Possui suporte ao SGBD da Oracle.

3. XML Spy

O XML Spy é um editor de XML e ambiente de desenvolvimento integrado, desenvolvido pela Altova, para modelagem, edição, transformação e depuração de tecnologias relacionadas ao XML. O XML Spy pode realizar a troca de dados entre um documento XML e o BD usando um mapeamento com base em um XML Schema. O acesso ao BD é feito através do ODBC.

3.2.3 Quadro comparativo

O Quadro 2 apresenta o resumo dos editores e IDE descritos anteriormente e apresenta uma breve comparação entre os mesmos.

Produto	**Desenvolvedor**	**Licença de uso**	**Tipo BD**	**BD para XML**	**XML para BD**
Unicenter SQL-Station	Computer Associates International	Comercial	Relacional	Sim	Sim
Stylus Studio	DataDirect Technologies	Comercial	Relacional, XML nativo	Sim	Sim
XML Spy	Altova	Comercial	Relacional	Sim	Sim

Quadro 2 – Comparação entre alguns dos principais editores e IDE para XML, que possibilitam a troca de dados entre documentos XML e BDs relacionais

3.3 Bancos de dados com suporte a XML

BDs com suporte a XML são BDs que oferecem extensões para troca de dados entre documentos XML e as suas próprias estruturas de dados.

São muitos os SGBDs no mercado que oferecem alguma extensão com suporte a XML. Tal suporte pode ser através de mapeamentos do esquema do documento XML para o esquema do BD e consequente carga do BD a partir dos dados do documento XML, porém com restrições no mapeamento e na estrutura dos documentos XML. Os documentos XML precisam estar de acordo com algum formato suportado, geralmente pouco flexível. Outra forma de suporte é o armazenamento de um documento XML

completo de algum jeito no BD. O armazenamento pode ser em colunas CLOB (*Character Large Object*), armazenamento como BLOB (*Binary Large Object*), armazenamento como texto puro ou algum formato binário proprietário, etc.
Como exemplos de alguns SGBD no mercado que possuem algum tipo de suporte para o XML temos o DB2, SQL Server, MySQL, PostGreSQL, Oracle, etc.

3.4 Softwares de Integração de Dados

Softwares de integração de dados, ou no inglês: *Data Integration Software*, são servidores autônomos projetados para fazer a troca de dados entre diversas fontes de dados, incluindo documentos XML e BDs. Geralmente são usados em processos como ETL (Extract, Transform, Load), EAI (Enterprise Application Integration), EII (Enterprise Information Integration) e frequentemente em aplicações SOA (Service-Oriented Architecture). (BOURRET; 2010).

Ainda de acordo com Bourret (2010), embora funcionem como middlewares, os mesmos se diferem nos seguintes pontos:

a) Múltiplas fontes de dados. Os Softwares de Integração de Dados são consideravelmente mais complexos e permitem fazer a transferência entre diversos formatos e fontes de dados, e não apenas entre documentos XML e BD. Bourret (2010) salienta o fato do número suportado de fontes de dados variar entre 10 e 300. O uso desses sistemas é consideravelmente mais difícil para os usuários do que os middlewares.
b) Desempenho. Os Softwares de Integração de Dados geralmente funcionam como servidores autônomos, porém alguns têm a capacidade de funcionar como bibliotecas locais. São projetados para ambientes de alto desempenho e normalmente possuem características como suporte a clusters, múltiplas threads e tolerância a faltas através de mecanismos de redundância.

Existem dezenas de sistemas de integração de dados e por serem softwares tão complexos, praticamente a totalidade dos mesmos possui licença comercial de uso. Entre os principais softwares de integração de dados disponíveis estão o Kettle, Oracle Data Integrator, Microsoft SQL Server 2008 Integration Services, DataMirror Transformation Server, Centerprise Data Integrator, etc.

4 DESENVOLVIMENTO DA FERRAMENTA

Foi desenvolvida uma ferramenta simples que permite fazer a carga de BDs relacionais diversos a partir de dados contidos em documentos XML. Por se tratar de uma ferramenta multi-plataforma, foi utilizada a linguagem Java na sua implementação, aproveitando suas várias facilidades, bem como sua vasta documentação.

No desenvolvimento da ferramenta foram utilizados padrões amplamente aceitos, como JAXP, SAX, DOM e XSLT para acesso e manipulação dos documentos XML, JDBC para acesso ao BD, além de XML Schema e DTD para o esquema dos documentos XML.

Neste capitulo a ferramenta é abordada em maiores detalhes. São descritas as suas funcionalidades, a sua arquitetura, detalhes da implementação e alguns testes realizados.

4.1 Arquitetura e Funcionalidades

A ferramenta desenvolvida agrega as seguintes funcionalidades:

a) Possibilita a carga de qualquer BD relacional cujo SGBD suporte a API JDBC, a partir de dados contidos em um documento XML;
b) Gera um arquivo XSL contendo comandos XSLT capazes de transformar o documento XML com os dados, em um script SQL com os comandos INSERT;
c) Disponibiliza uma linguagem simples para mapeamento XML-Relacional, com a qual é possível criar arquivos de mapeamento (arquivo .map);
d) Gera automaticamente o mapeamento XML-Relacional a partir de esquemas XML (DTD e XML Schema);
e) Gera automaticamente comandos CREATE TABLE para as tabelas envolvidas na transferência de dados;
f) Dá suporte a namespaces, geração automática de chaves únicas e transações para SGBD que as suportam.

A classe principal do sistema é a classe Mapeamento. Um objeto desta classe pode ser visto como a representação interna de um arquivo de mapeamento (arquivo XML com a descrição do mapeamento usando a linguagem de mapeamento oferecida pela ferramenta) e contém todas as informações sobre o mapeamento (XML-Relaconal) de cada elemento no documento XML para o BD, o que inclui qual a tabela a ser mapeada, a coluna mapeada para cada propriedade do elemento, etc.

Esse objeto é então usado na carga do BD, juntamente com o documento XML que contém os dados.

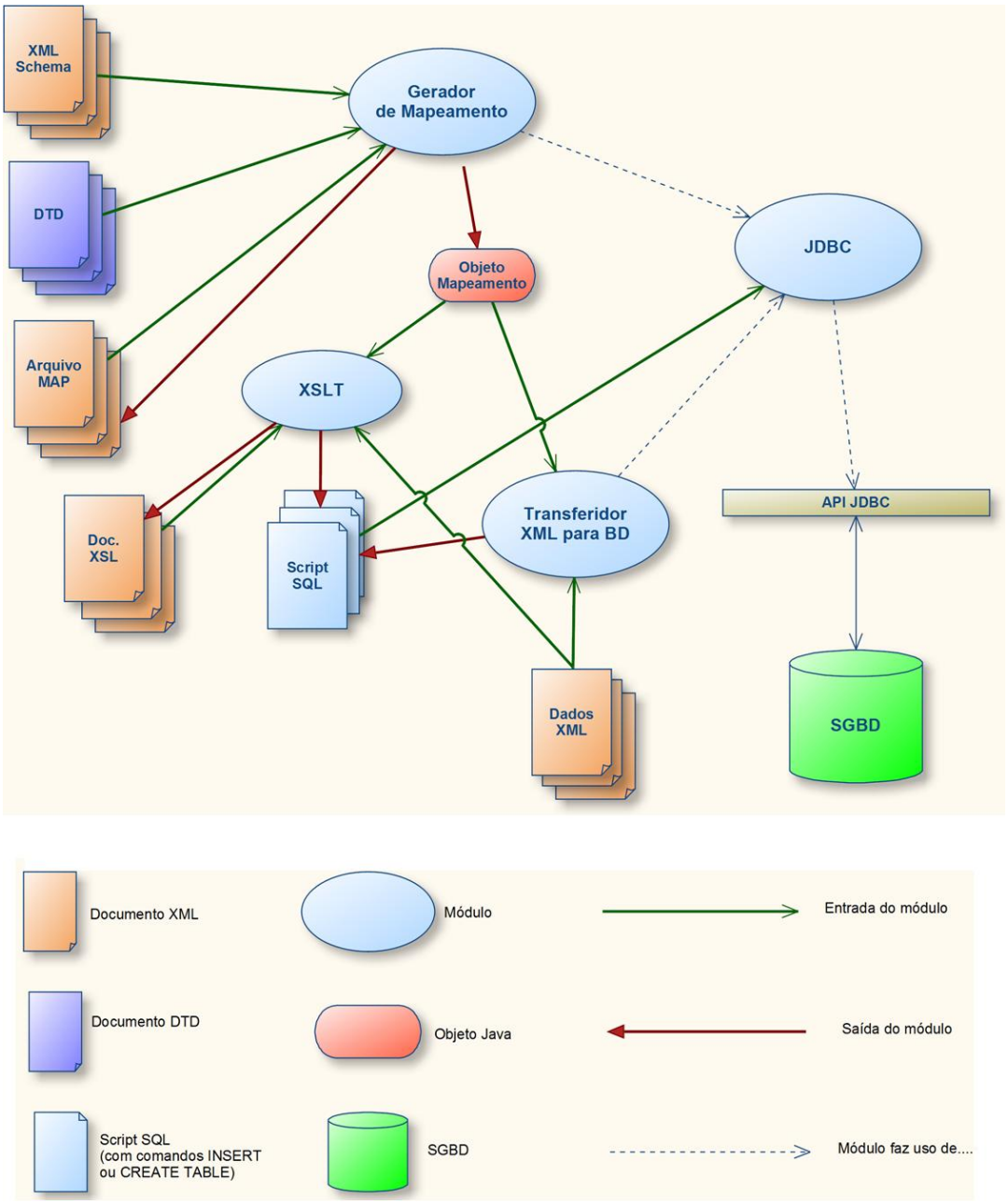

Figura 1: Arquitetura da ferramenta desenvolvida na visão de módulos

Conforme ilustrado na Figura 1, são quatro os módulos desenvolvidos:

a) Módulo gerador de mapeamento;
b) Módulo XSLT;
c) Módulo transferidor de dados XML para o BD;
d) Modulo JDBC.

O módulo **gerador de mapeamento** é o módulo responsável por processar os arquivos de mapeamento (arquivos .map) e criar objetos da classe Mapeamento a partir de tais arquivos. O módulo ainda tem a capacidade de processar esquemas XML (DTD e XML Schema) e a partir destes, gerar arquivos de mapeamento. Os arquivos gerados podem, então, ser editados pelo usuário e submetidos novamente ao módulo gerador, para a criação de objetos da classe Mapeamento. Caso seja especificado pelo usuário, o módulo gerador faz uso do módulo JDBC para validar o mapeamento gerado, verificando a existência das tabelas e colunas especificadas no mapeamento, no BD, bem como o tipo de dado das colunas, etc.

O objeto da classe Mapeamento gerado pelo módulo gerador de mapeamento, serve de entrada tanto para o módulo XSLT como para o módulo transferidor de dados.

O **módulo XSLT** é o módulo responsável por gerar, a partir do objeto da classe Mapeamento recebido como entrada, arquivos XSL com comandos XSLT capazes de transformar o documento XML, que contém os dados, em um script SQL com comandos INSERT necessários para fazer a carga do BD. Este módulo tem ainda a capacidade de executar a transformação e gerar scripts SQL a partir de arquivos XSL já gerados e dos documentos XML com os dados.

O **módulo transferidor de dados XML para BD** é o módulo responsável por fazer a carga do BD a partir dos dados contidos em documentos XML. Recebendo como entrada um objeto da classe Mapeamento, gerado pelo módulo gerador de mapeamentos e um documento XML com os dados, este último é então processado e, a partir dos dados contidos no mesmo, é feita a carga do BD. Este módulo tem ainda a capacidade de gerar scripts SQL com os comandos INSERT correspondentes ou scripts SQL com comandos CREATE TABLE para criar as tabelas no BD necessárias para a carga do mesmo. O módulo transferidor faz uso do módulo JDBC para enviar os comandos SQL para o SGBD e para obter informações do BD.

O módulo JDBC é o módulo responsável pela comunicação com o BD através da API JDBC. Outros módulos fazem uso deste módulo para enviar comandos SQL

para o SGBD ou para obter alguma informação do BD. O módulo JDBC ainda tem a capacidade de receber arquivos SQL com comandos INSERT ou CREATE TABLE e enviá-los diretamente para o SGBD.

O funcionamento básico da ferramenta consiste em receber como entradas o documento XML com os dados e o arquivo de mapeamento (.map) com as regras do mapeamento, escritas usando a linguagem de mapeamento. Tal arquivo é então passado ao módulo gerador de mapeamento onde é processado por uma das classes geradoras e um objeto Mapeamento é gerado. Esse objeto pode ser serializado para uso posterior ou passado, juntamente com o(s) documentos(s) XML, ao módulo transferidor de dados, que se encarrega de fazer a conversão dos dados de acordo com o objeto Mapeamento recebido e a posterior inserção dos mesmos no BD relacional. O transferidor de dados ainda tem a capacidade de gerar automaticamente chaves únicas caso seja necessário no momento da inserção no BD.

Além disso, é possível gerar um script SQL com os comandos CREATE TABLE que pode ser usado para gerar as tabelas envolvidas na transferência de dados. Note-se que um mesmo objeto Mapeamento pode ser utilizado mais do que uma vez para a carga do BD e para gerar o esquema relacional.

Outra entrada possível para o sistema é um arquivo DTD ou XML Schema contendo o esquema do documento XML. Neste caso, um objeto Mapeamento é gerado automaticamente pelas classes geradoras e então se pode gerar o esquema relacional, salvar o mapeamento no sistema de arquivos (em forma de um arquivo .map) ou fazer a carga do BD a partir de dados contidos em um determinado documento XML.

O arquivo XSD ou DTD pode ser o arquivo de validação para o documento XML que contém os dados ou ainda pode ser obtido com auxílio de ferramentas como o db2xsd, que são capazes de gerar um XML Schema ou DTD a partir de um esquema relacional ou de um documento XML.

No seguinte exemplo é mostrado um XML Schema simples, que declara um elemento complexo de nome ‘recado’:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="recado">
```

```
      <xs:complexType>
        <xs:sequence>

          <xs:element name="de" type="xs:string"/>
          <xs:element name="para" type="xs:string"/>
          <xs:element name="mensagem" type="xs:string"/>

        </xs:sequence>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Recebendo este arquivo como entrada, um objeto Mapeamento seria gerado e o elemento complexo ‘recado’ seria mapeado para a tabela ‘Recado’ e os elementos simples ‘de’, ‘para’ e ‘mensagem’ seriam mapeados para as colunas ‘De’, ‘Para’ e ‘Mensagem’ da tabela ‘Recado’. Evidentemente que para casos em que elementos complexos contenham outros elementos complexos, tais elementos também são mapeados para tabelas e o relacionamento entre as tabelas é definido. A linguagem de mapeamento oferecida pela ferramenta e o mapeamento dos esquemas XML (DTD e XML Schema) para o esquema relacional são descritos em maiores detalhes no decorrer deste capítulo.

O sistema pode ainda gerar um arquivo XSL a partir do mapeamento. O arquivo contém comandos XSLT, e quando executado por um transformador XSLT, sobre o documento XML, gera como saída um script SQL com os comandos INSERT. O arquivo XSL gerado pode ser usado em outras aplicações ou processado pelo próprio módulo XSLT da ferramenta.

Por fim, os scripts SQL gerados tanto pelo módulo XSLT como pelo módulo transferidor de dados podem ser processados diretamente pelo módulo JDBC da ferramenta e os comandos INSERT ou CREATE TABLE enviados para o SGBD.

4.2 O mapeamento XML-Relacional

O modelo de dados de um documento XML é hierárquico. Isto significa que os dados estão organizados numa estrutura hierárquica de árvore, o que difere totalmente do modelo relacional, que se baseia em tabelas e relacionamento entre as mesmas através do uso de chaves. Portanto, para que seja possível fazer a carga de um BD relacional a partir de um documento XML, torna-se necessário fazer a conversão da estrutura hierárquica para a estrutura de tabelas.

Com esta finalidade, é oferecida uma linguagem baseada em XML que possibilita descrever o mapeamento dos dados XML para as tabelas do BD relacional.

Já os esquemas XML (DTD e XML Schema) podem ser mapeados automaticamente pela ferramenta, para o esquema relacional, gerando arquivos de mapeamento que podem então ser editados pelo usuário.

Na ferramenta proposta, o mapeamento XML-relacional é feito através de dois conceitos importantes: **entidade** e **propriedade.** Nesta abordagem, os elementos de um documento XML podem ser vistos tanto como entidades como propriedades. Os elementos complexos, ou seja, elementos que contêm outros elementos são geralmente vistos como entidades e posteriormente serão mapeados para tabelas do BD relacional. Já os elementos simples de texto ou do tipo PCDATA (*Parsed Character Data*) são vistos como propriedades de alguma entidade. Quando um elemento de um documento XML é visto como uma propriedade, tal propriedade pertence à entidade correspondente do seu elemento pai (ascendente direto) e será mapeado para uma coluna da tabela correspondente.

Além disto, quando um elemento é considerado uma entidade, os seus atributos e elementos texto simples são vistos como propriedades da entidade em questão. Vale ressaltar que uma propriedade não pode conter outras propriedades. Como consequência direta disto, quando um elemento é visto como propriedade, os seus PCDATA são vistos como parte do seu próprio valor e seus atributos são vistos como propriedades da entidade do elemento pai. Apesar de o mapeamento poder ser gerado automaticamente, o usuário pode modificá-lo se assim for necessário.

Elementos de um documento XML usualmente contêm outros elementos. O relacionamento entre um elemento e seu elemento filho (descendente) é classificado dependendo de como são vistos os elementos envolvidos. São três as possibilidades:

a) Relacionamento entre-entidades, quando ambos os elementos envolvidos são vistos como entidades;
b) Relacionamento entidade-propriedade, quando o elemento pai é visto como uma entidade e o elemento filho como sua propriedade;
c) Relacionamento entre-propriedades, quando ambos os elementos são vistos como propriedades.

Através da linguagem de mapeamento oferecida pela ferramenta, é possível especificar como será visto cada elemento do documento XML e o relacionamento entre os mesmos. No exemplo a seguir, o elemento ‘Funcionario’ e ‘Departamento’ são elementos complexos e como tal são vistos como entidades. O atributo ‘matricula’ e os elementos simples ‘nomeF’ e ‘anoNasc’ são todos vistos como propriedades da entidade ‘Funcionario’. O atributo ‘num’ e o elemento ‘nomeD’ são vistos como propriedades da entidade ‘Departamento’.

Por fim, o relacionamento entre o elemento ‘Funcionario’ e ‘Departamento’ é visto como sendo um relacionamento entre-entidades.

```
<Funcionario matricula = 01293>
        <nomeF>Manuel Silva</nomeF>
        <anoNasc>1980</anoNasc>
        <Departamento num = 7>
                <nomeD>Comercial</nomeD>
                ...
        </Departamento>
        ...
</Funcionario>
```

Uma vez definidas as entidades envolvidas, o mapeamento se torna direto, sendo que cada entidade corresponde a uma tabela, as suas propriedades correspondem às colunas da tabela e os relacionamentos entre-entidades correspondem a pares de chaves (única e estrangeira) ou tabelas associativas. O mapeamento através de tabela associativa é explicado mais adiante neste capítulo.

4.2.1 A linguagem de mapeamento XML-Relacional

Conforme citado anteriormente, a ferramenta disponibiliza uma linguagem simples baseada em XML. A linguagem oferecida é um subconjunto modificado e simplificado da linguagem proposta por BOURRET (2004). A linguagem não é mais do que um esquema XML com descrição de tags bem específicas e com significados bem definidos. Um arquivo de mapeamento (arquivo MAP) é um documento XML bem formado e que respeita as regras da linguagem descritas pelo esquema XML. O esquema XML com a linguagem de mapeamento é então usado para validar os arquivos MAP, antes de o sistema gerar o objeto Mapeamento correspondente. A seguir é mostrado o formato do arquivo de mapeamento de forma simplificada e em seguida cada tag da linguagem XML é explicada em maiores detalhes. Por fim, é dado um exemplo completo do mapeamento de um documento XML para BD.

```
<MapXMLRel>
    <Opcoes>
        ...
    </Opcoes>
    <Mapeamentos>
        ...
        < Entidade >
        < /Entidade>
        ...
    </Mapeamentos >
</MapXMLRel>
```

O elemento ‘MapXMLRel’ é o elemento raiz do arquivo de mapeamento, ou seja, todos os outros elementos são descendentes do mesmo.

O elemento ‘Opcoes’ é opcional e serve para especificar algumas opções como o uso de namespaces e como as strings vazias no documento XML devem ser tratadas. Contém o elemento ‘NameSpace’ e o ‘TrataStringVaziaComoNulo’. A estrutura do elemento é a seguinte:

```
<Opcoes>
    < TrataStringVaziaComoNulo/>
    <Namespace Prefixo=”algumPrefixo” URI=”algumaURI”/>
    <Namespace Prefixo=”algumPrefixo2” URI=”algumaURI2”/>
    ...
```

```
        <Namespace Prefixo=”algumPrefixoN” URI=”algumaURIN”/>
    </Opcoes>
```

Tanto o elemento ‘TrataStringVaziaComoNulo’ como o ‘Namespace’ são opcionais. Se o elemento ‘TrataStringVaziaComoNulo’ estiver presente, as strings vazias encontradas no documento XML que contém os dados, são tratadas como nulas (NULL), caso contrário são tratadas como strings quaisquer. No exemplo a seguir, o valor do atributo ‘ano’ é uma string vazia:

```
<livro autor=”João” ano=””/>
```

O elemento ‘NameSpace’ pode ou não aparecer e pode se repetir N vezes. Os seus dois atributos ‘Prefixo’ e ‘URI’ indicam o prefixo de um namespace que será usado no documento XML com os dados e a URI do namespace, respectivamente. Este elemento é usado na geração de scripts, como o XSLT.

O elemento ‘Mapeamentos’ é o elemento que contém um ou mais elementos ‘Entidade’, que por sua vez contém a informação de como mapear um determinado elemento do documento XML visto como entidade para o BD relacional. Além disso, o elemento ‘Mapeamentos’ pode conter um elemento opcional chamado ‘IgnoraElementoRaiz’. A sua estrutura é a seguinte:

```
<Mapeamentos>
        <IgnoraElementoRaiz>
                ...
        </IgnoraElementoRaiz>
        <Entidade>
                ...
        </Entidade>
        ...
</Mapeamentos >
```

O Elemento ‘IgnoraElementoRaiz’ permite especificar que o elemento raiz do documento XML deve ser ignorado no mapeamento e na transferência de dados. A estrutura do XML exige que exista um elemento raiz em cada documento. Porém, tal elemento pode existir apenas para suprir essa exigência e não ter qualquer relação com o BD relacional. Por exemplo, ao criar um documento XML que contenha informações de

pessoas a serem cadastradas no BD, suponhamos que a estrutura do documento seja a seguinte:

```
<Pessoas>
        <Pessoa>
                <nome>Marcelo Dias<nome/>
                ...
        </Pessoa>
        <Pessoa>
                ...
        </Pessoa>
        <Pessoa>
                ...
        </Pessoa>
        ...
</Pessoas>
```

Neste caso, o elemento ‘Pessoas’ existe apenas para cumprir o requisito de elemento raiz, porém não tem qualquer significado para o BD. Não será mapeado para nenhuma tabela. Como tal, deve ser ignorado e o elemento ‘Pessoa’ deve ser processado. O elemento ‘IgnoraElementoRaiz’ é opcional e pode aparecer várias vezes visto que podemos ter um documento com várias instâncias sem nenhuma relação. O elemento filho ‘FilhoRaiz’ permite especificar quais os elementos filhos da raiz serão mapeados para o BD e pode aparecer várias vezes. A estrutura é:

```
<IgnoraElementoRaiz>
        <Elemento nome=”Pessoas”/>
        < FilhoRaiz >
                <Elemento nome=”Pessoa”/>
        < /FilhoRaiz>
</IgnoraElementoRaiz>
```

O elemento ‘Entidade’ é o elemento que contém as informações do mapeamento de cada elemento do documento XML visto como entidade, para o BD relacional. Ele permite indicar para qual tabela o elemento será mapeado e especificar o mapeamento das suas propriedades e relacionamentos com outros elementos. A estrutura é:

```
<Entidade>
        <Elemento nome=”nomeElemento”/>
        <Tabela nome=”NOMETABELA”/>
        <Propriedade>
```

```
            ...
        </Propriedade>
        <EntidadeRelacionada>
            ...
        </EntidadeRelacionada>
        <IgnoraElemento>
            ...
        </IgnoraElemento>
        ...
</Entidade>
```

O elemento de nome ‘Elemento’ especifica qual o elemento a ser mapeado como entidade. O atributo ‘nome’ do elemento ‘Tabela’ especifica o nome da tabela no BD para a qual a entidade será mapeada. O elemento ‘Propriedade’ que é opcional e pode se repetir várias vezes especifica quais atributos e elementos simples serão mapeados para quais colunas da tabela correspondente à entidade sendo mapeada. O atributo opcional ‘vazio’ do elemento ‘Elemento’ indica se trata-se de um elemento vazio e por default o seu valor é “N”. Caso o valor seja “S”, a propriedade é mapeada para uma coluna de tipo *BOOLEAN*. Se o elemento estiver presente no documento XML, o valor da coluna será *true*, caso contrário, *false.*

A estrutura é a seguinte:

```
<Propriedade>
        <Atributo nome=”nomeAtributo” />
        <!— ou  <Elemento nome=”nomeElemento” vazio=”S ou N”/> -->
        <Coluna nome =”nomeColuna”/>
</Propriedade>
```

O elemento ‘EntidadeRelacionada’ que também é opcional e pode aparecer várias vezes especifica os relacionamentos da entidade mapeada com outras entidades, bem como as chaves única e estrangeira utilizadas no BD para a junção das tabelas correspondentes ou ainda a tabela associativa utilizada no BD para relacionar as tabelas. O elemento ‘EntidadeRelacionada’ possui um atributo obrigatório de nome ‘tipoRelacionamento’ que permite especificar o tipo de relacionamento entre as duas entidades (‘Direto’ ou ‘N-N’). O tipo ‘Direto’ representa os relacionamentos ‘1-1’ e ‘1-N’ pois os mesmos são tratados de forma igual. Ambos são relacionamentos em que uma das tabelas envolvidas armazena uma chave estrangeira que referencia uma entidade da outra tabela. Nestes casos a estrutura é a seguinte:

```
<EntidadeRelacionada tipoRelacionamento=”Direto”>
        <Elemento nome=”nomeElemento”/>
        <ChavesJuncao ChaveEntidadePai=”Unica”>
                <ChaveUnica gerarAutomaticamente=”N”>
                        <Coluna nome =”nomeColuna”/>
                </ChaveUnica>
                <ChaveEstrangeira>
                        <Coluna nome =”nomeColuna”/>
                </ChaveEstrangeira>
        </ChavesJuncao>
</EntidadeRelacionada>
```

O elemento ‘Elemento’ especifica qual o elemento que se relaciona com o elemento sendo mapeado. O atributo ‘ChaveEntidadePai’ indica se a chave presente na tabela correspondente à entidade pai (lembrando que as declarações de entidades relacionadas estão aninhadas dentro da declaração de uma entidade) é única ou estrangeira. Isso possibilita mais flexibilidade na formação dos documentos XML. A mesma informação pode ser estruturada de formas diferentes. No exemplo seguinte, o elemento pai é o elemento ‘Funcionario’ e um dos elementos filhos é o elemento ‘Departamento’. Considerando que é a tabela ‘Funcionario’ que armazena uma chave estrangeira (única na tabela ‘Departamento’) que aponta para o departamento a qual o funcionário pertence, no mapeamento da entidade ‘Funcionario’, a entidade relacionada ‘Departamento’ seria mapeada na sessão <EntidadeRelacionada> e o atributo ‘ChaveEntidadePai’ teria seu valor como ‘Estrangeira’.

```
<Funcionario>
        <Nome>Gilberto Braga</Nome>
        <AnoNasc>1987</AnoNasc>
        <Departamento num=”1”>
                < Nome> Financeiro<Nome/>
        </Departamento>
</Funcionario>
```

Supondo nomes para as tabelas e colunas, o mapeamento da entidade Funcionario seria o seguinte:

```
<Entidade>
        <Elemento nome=”Funcionario”/>
        <Tabela nome=”TFuncionario”/>
        <Propriedade>
                <Elemento nome=”AnoNasc”/>
                <Coluna nome =”anoNasc”/>
        </Propriedade>
```

```
        <Propriedade>
                <Elemento nome=”Nome”/>
                <Coluna nome =”nome”/>
        </Propriedade>

        <EntidadeRelacionada tipoRelacionamento=”Direto”>
                <Elemento nome=”Departamento”/>
                <ChavesJuncao ChaveEntidadePai=”Estrangeira”>
                        <ChaveUnica gerarAutomaticamente=”N”>
                                <Coluna nome =”numero”/>
                        </ChaveUnica>
                        <ChaveEstrangeira>
                                <Coluna nome =”num_dep”/>
                        </ChaveEstrangeira>
                </ChavesJuncao>
        </EntidadeRelacionada>
</Entidade>
```

A mesma informação pode ser organizada de forma diferente no documento XML. Por exemplo:

```
<Departamento num=”1”>
                <Nome> Financeiro</Nome>
                <Funcionario>
                        <Nome>Gilberto Braga</Nome>
                        <AnoNasc>1987</AnoNasc>
                </Funcionario>
                <Funcionario>
                        ...
                </Funcionario>
                ...
</Departamento>
```

Neste caso, no mapeamento da entidade ‘Departamento’, a entidade relacionada ‘Funcionario’ seria mapeada na sessão <EntidadeRelacionada> e o atributo ‘ChaveEntidadePai’ teria seu valor como “Unica”. Supondo nomes para as tabelas e colunas, o mapeamento da entidade Departamento seria o seguinte:

```
<Entidade>
        <Elemento nome=”Departamento”/>
        <Tabela nome=”TDepartamento”/>
        <Propriedade>
                <Atributo nome=”num”/>
                <Coluna nome =”numero”/>
        </Propriedade>
        <Propriedade>
                <Elemento nome=”Nome”/>
                <Coluna nome =”nome”/>
```

```
        </Propriedade>

        <EntidadeRelacionada tipoRelacionamento=”Direto”>
                <Elemento nome=”Funcionario”/>
                <ChavesJuncao ChaveEntidadePai=”Unica”>
                        <ChaveUnica gerarAutomaticamente=”N”>
                                <Coluna nome =”num”/>
                        </ChaveUnica>
                        <ChaveEstrangeira>
                                <Coluna nome =”num_dep”/>
                        </ChaveEstrangeira>
                </ChavesJuncao>
        </EntidadeRelacionada>
</Entidade>
```

Os elementos ‘ChaveEstrangeira’ e ‘ChaveUnica’ permitem especificar as chaves usadas no BD para fazer a junção das tabelas. O elemento ‘Coluna’ pode se repetir mais de uma vez caso as chaves sejam compostas por mais de uma coluna. O atributo ‘gerarAutomaticamente’ indica se a chave única deve ser gerada pela ferramenta.

Já em caso de relacionamento “N-N”, em vez do elemento ‘ChavesJuncao’, está presente o elemento ‘TabelaAssociativa’, que especifica detalhes da tabela associativa usada no banco para o relacionamento N-N. A estrutura é a seguinte:

```
<EntidadeRelacionada tipoRelacionamento=”N-N”>
        <Elemento nome=”nomeElemento”/>
        <TabelaAssociativa nome=”NOMETABELAASSOC” elemento=”nelemento
idref=”atributoIDREF” idrefs=”atributoIDREFS”/>
                <Coluna nome=”COLUNA1” valor=”TabPai.Coluna”/>
                <Coluna nome=”COLUNA2” valor=”TabFilho.Coluna”/>
                <Coluna .../>
                ...
        </ TabelaAssociativa>
</EntidadeRelacionada>
```

O elemento ‘Elemento’ especifica o elemento com o qual a entidade sendo mapeada se relaciona através de uma tabela associativa.

O atributo ‘nome’ do elemento ‘TabelaAssociativa’ especifica o nome da tabela associativa no BD. O atributo ‘elemento’ é opcional e pode especificar qual o elemento simbólico no documento XML que contem as informações que serão mapeadas para a tabela associativa através de uma expressão XPath. Os atributos idref e idrefs são

opcionais e especificam que atributo de valor único no documento XML aponta para o elemento sendo mapeado na seção ‘EntidadeRelacionada’.

No exemplo a seguir são mostradas duas formas distintas de mapear o relacionamento N-N entre as entidades ‘Projeto’ e ‘Empregado’. Seja um documento XML com a seguinte estrutura:

```
<empregados>
        <empregado id=”m-121”>
                <nome>Roberto Silva</nome>
                ...
        </empregado>
        ...
</empregados>
<projetos>
        <projeto id=”1” nome=”p1” emps=”m-121  m-34  m-444  m-23  m-123”/>
        <projeto id=”2” nome=”p2” emps=”m-121  m-44”/>
        ...
</projetos>
```

Suponha que as tabelas mapeadas foram TProjeto e TEmpregado e as colunas Pid e Eid são as colunas referentes às chave primárias nas tabelas, respectivamente. No mapeamento da entidade ‘Projeto’, a seção ‘EntidadeRelacionada’ seria a seguinte:

```
<EntidadeRelacionada tipoRelacionamento=”N-N”>
        <Elemento nome=”empregado”/>
        <TabelaAssociativa nome=”PROJ_EMP” idrefs=”emps”/>
                <Coluna nome=”ID_Proj” valor=”TProjeto.Pid”/>
                <Coluna nome=”ID_Emp” valor=”TEmpregado.Eid”/>
        </ TabelaAssociativa>
</EntidadeRelacionada>
```

Outra possível alternativa seria:

```
<empregados>
        <empregado id=”m-121”>
                <nome>Roberto Silva</nome>
                ...
        <empregado>
        ...
</empregados>
<projetos>
        <projeto id=”1” nome=”p1” />
                <emps>
                        <empref  eid=”m-121”>
                        <empref  eid=”m-34”>
                        ...
```

```
            </emps>
        </projeto>
        ...
</projetos>

<EntidadeRelacionada tipoRelacionamento=”N-N”>
    <Elemento nome=”empregado”/>
    <TabelaAssociativa nome=”PROJ_EMP” elemento=”emps/empref” idref=”eid”/>
        <Coluna nome=”ID_Proj” valor=”TProjeto.Pid”/>
        <Coluna nome=”ID_Emp” valor=”TEmpregado.Eid”/>
    </ TabelaAssociativa>
</EntidadeRelacionada>
```

Por fim, o elemento ‘<IgnoraElemento>’ serve para informar que um determinado elemento no documento XML existe para fins de organização dos dados mas não deve ser mapeado para uma tabela do BD. Neste caso, o elemento é ignorado e suas propriedades são mapeadas como sendo propriedades da entidade pai. Por exemplo:

```
<Funcionario>
        <Nome>Gilberto Braga</Nome>
        <AnoNasc>1987</AnoNasc>
        <Endereco>
                < Rua> Antonio Magalhaes</Rua>
                <Numero>123</Numero>
                <Cidade>Florianopolis</Cidade>
        </Endereco>
        ...
</Funcionario>
```

Supondo que a informação de endereço, em vez de armazenada numa tabela separada, esteja na própria tabela ‘Funcionario’, no mapeamento da entidade ‘Funcionario’, o elemento ‘Endereco’ seria declarado na tag <IgnoraElemento>.

O exemplo a seguir apresenta um documento XML simples e em seguida apresenta o mapeamento para o mesmo. O documento pode conter informações de N funcionários:

```
<Funcionarios>
        <Funcionario Matricula=”07363”>
                <Nome>Maria Silva</Nome>
                <Sexo>F</Sexo>
                <Departamento num=”7”>
                        < Nome> Comercial</Nome>
                </Departamento>
        </Funcionario>
        <Funcionario>
```

```
            ...
        </Funcionario>
        ...
</Funcionarios>
```

Considerando as duas tabelas no BD:

a) FUNCIONARIO (matr, nome, sexo, numDep)
b) DEPARTAMENTO (numero, nomeD)

Desconsiderando a *tag* <Opcoes> que é opcional, o arquivo de mapeamento seria o seguinte:

```
<?xml version="1.0" ?>
<MapXMLRel>
        <Mapeamentos>
                <IgnoraElementoRaiz>
                        <Elemento nome=”Funcionarios”/>
                        < FilhoRaiz >
                                <Elemento nome=”Funcionario”/>
                        < /FilhoRaiz>
                </IgnoraElementoRaiz>
                <Entidade>
                        <Elemento nome=”Funcionario”/>
                        <Tabela nome=”FUNCIONARIOS”/>
                        <Propriedade>
                                <Atributo nome=” Matricula”/>
                                <Coluna nome =”matr”/>
                        </Propriedade>
                        <Propriedade>
                                <Elemento nome=”Nome”/>
                                <Coluna nome=”nome”/>
                        </Propriedade>
                        <Propriedade>
                                < Elemento nome=”Sexo”/>
                                <Coluna nome =”sexo”/>
                        </Propriedade>
                        <EntidadeRelacionada tipoRelacionamento=”Direto”>
                                <Elemento nome=”Departamento”/>
                                <ChavesJuncao ChaveEntidadePai=”Estrangeira”>
                                        <ChaveUnica gerarAutomaticamente=”N”>
                                                <Coluna nome =”numero”/>
                                        </ChaveUnica>
                                        <ChaveEstrangeira>
                                                <Coluna nome =”numDep”/>
                                        < /ChaveEstrangeira>
                                </ChavesJuncao>
                        </EntidadeRelacionada>
                < /Entidade>
                < Entidade>
                        <Elemento nome=”Departamento”/>
                        <Tabela nome=”DEPARTAMENTO”/>
                        <Propriedade>
                                <Atributo nome=”num”/>
                                <Coluna nome =”numero”/>
                        </Propriedade>
                        <Propriedade>
                                <Elemento nome=”Nome”/>
                                <Coluna nome =”nomeD”/>
                        </Propriedade>
                </Entidade>
        </Mapeamentos>
</MapXMLRel>
```

Note-se que a ferramenta consegue gerar o arquivo de mapeamento automaticamente neste formato, a partir de um objeto Mapeamento já gerado ou a partir de um DTD ou XML Schema. O arquivo pode então ser editado manualmente pelo usuário.

4.2.2 Mapeamento de esquemas XML para o esquema relacional

Para mapear um esquema XML para o esquema relacional, também é utilizado o conceito de entidade e propriedade.

Cada elemento complexo, ou seja, que possua elementos filhos ou atributos, é mapeado como entidade e, consequentemente para uma tabela no BD. Os atributos e elementos filho simples são considerados propriedades e mapeados como colunas na tabela do elemento pai correspondente. Caso a referência para um elemento filho simples ou para um atributo seja opcional, a coluna gerada pode assumir valores nulos.

Uma coluna é gerada para a chave primária, e uma coluna para a chave estrangeira caso o elemento possua um elemento pai.

O processo de mapeamento é feito em 3 passos:

a) No primeiro passo, são geradas tabelas para elementos complexos e uma chave primária para as mesmas;
b) No segundo passo, são geradas colunas para os elementos filhos simples;
c) No terceiro passo, são geradas chaves estrangeiras para referências a elementos complexos. Caso o elemento complexo possua um elemento pai, uma coluna com chave estrangeira é gerada na sua tabela correspondente.

No processo de mapeamento algumas convenções de nomes são assumidas. A tabela gerada a partir de um elemento complexo tem o mesmo nome que o elemento. As colunas geradas para cada atributo e elemento simples têm o mesmo nome que os mesmos. A coluna para a chave primária tem o mesmo nome da tabela seguido de “PK”. A coluna para a chave estrangeira tem o mesmo nome da tabela que é referenciada, seguido de “FK”.

Os atributos definidos como ID/IDREF ou key/keyref são mapeados como chave primária/estrangeira para uma coluna de mesmo nome dos atributos. Caso não exista nenhum atributo assim mapeado, uma chave primária é gerada com o nome da tabela seguido de “PK”.

A ferramenta tem a capacidade de gerar o mapeamento a partir de esquemas XML (DTD e XML Schemas simples). Isso é interessante, pois o esquema de validação dos documentos XML contendo os dados pode ser usado para gerar o arquivo de mapeamento.

Outra opção é fazer o uso de ferramentas que geram um DTD ou XML Schema, a partir de um documento XML. Existem várias ferramentas *online*, como por exemplo, o conjunto de utilitários XML da HiT Software.

Assim, o DTD e o XML Schema podem ser utilizados pela ferramenta para gerar o arquivo de mapeamento automaticamente, e o usuário pode editá-lo se for preciso.

Embora a linguagem de mapeamento seja simples, para documentos XML com muitos elementos, a escrita manual do arquivo de mapeamento pode se tornar um pouco cansativa. Sendo assim, os esquemas XML podem ser usados para facilitar o processo, pelo menos como ponto de partida para geração do arquivo de mapeamento.

4.3 Implementação

O código da ferramenta encontra-se dividido em quatro pacotes Java:

a) **Mapeamento**: contendo todas as classes envolvidas no mapeamento XML-Relacional;

b) **Geradores**: contendo as classes utilizadas para gerar o objeto Mapeamento a partir de um arquivo de mapeamento, um DTD ou um XML Schema;

c) **Transferidor**: contendo as classes necessárias para fazer a carga do BD, bem como para gerar o esquema relacional e gerar o arquivo XSL;

d) **Utils**: contendo utilitários e classes de uso geral

A Figura 2 ilustra os 4 pacotes citados e as principais classes pertencentes a cada um.

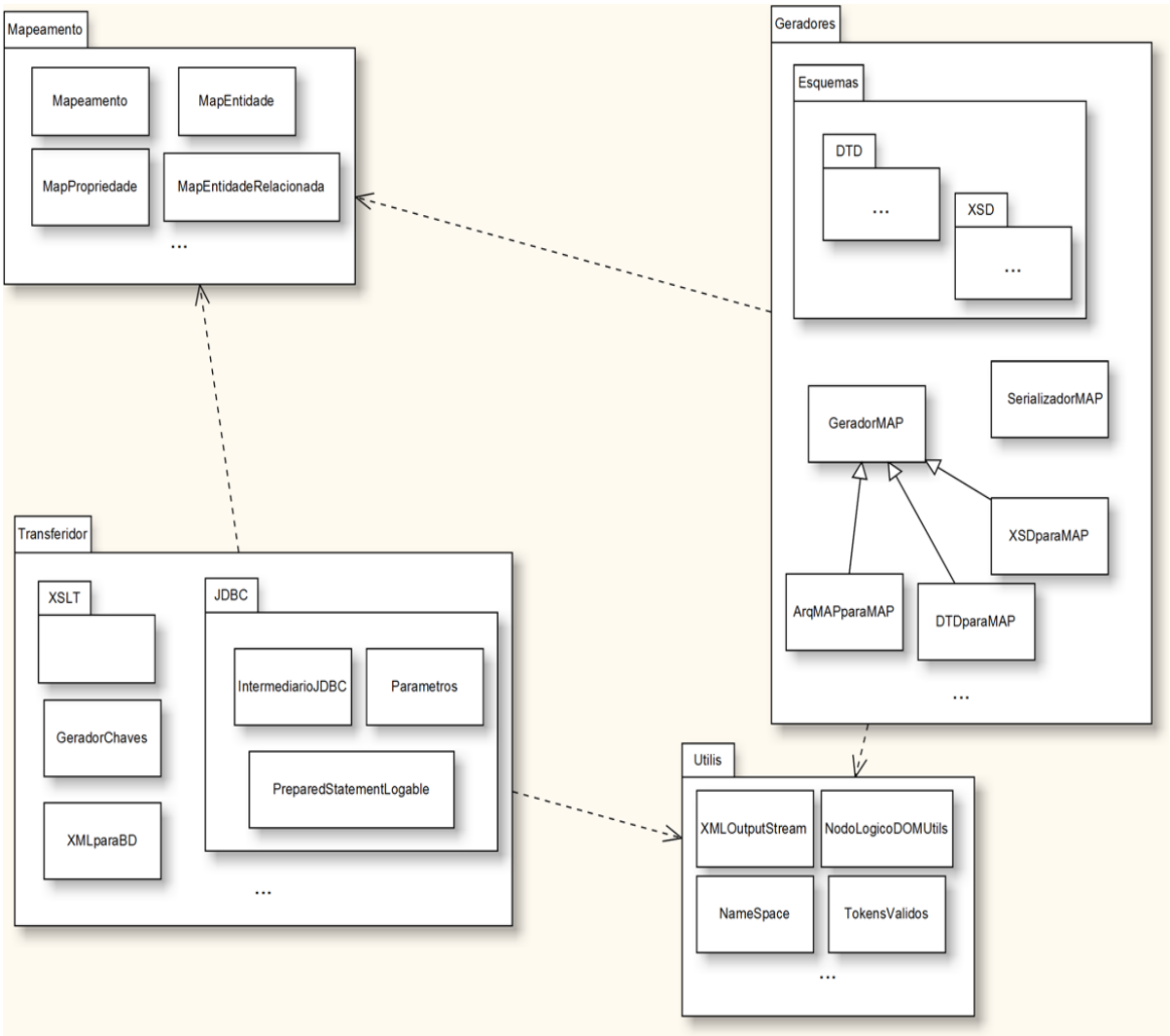

Figura 2: diagrama de pacotes

Na Figura 3, é apresentado um diagrama com as principais classes do sistema. Devido à grande quantidade de classes, apenas as principais e de uso geral são apresentadas. As demais são classes auxiliares e de uso restrito da implementação.

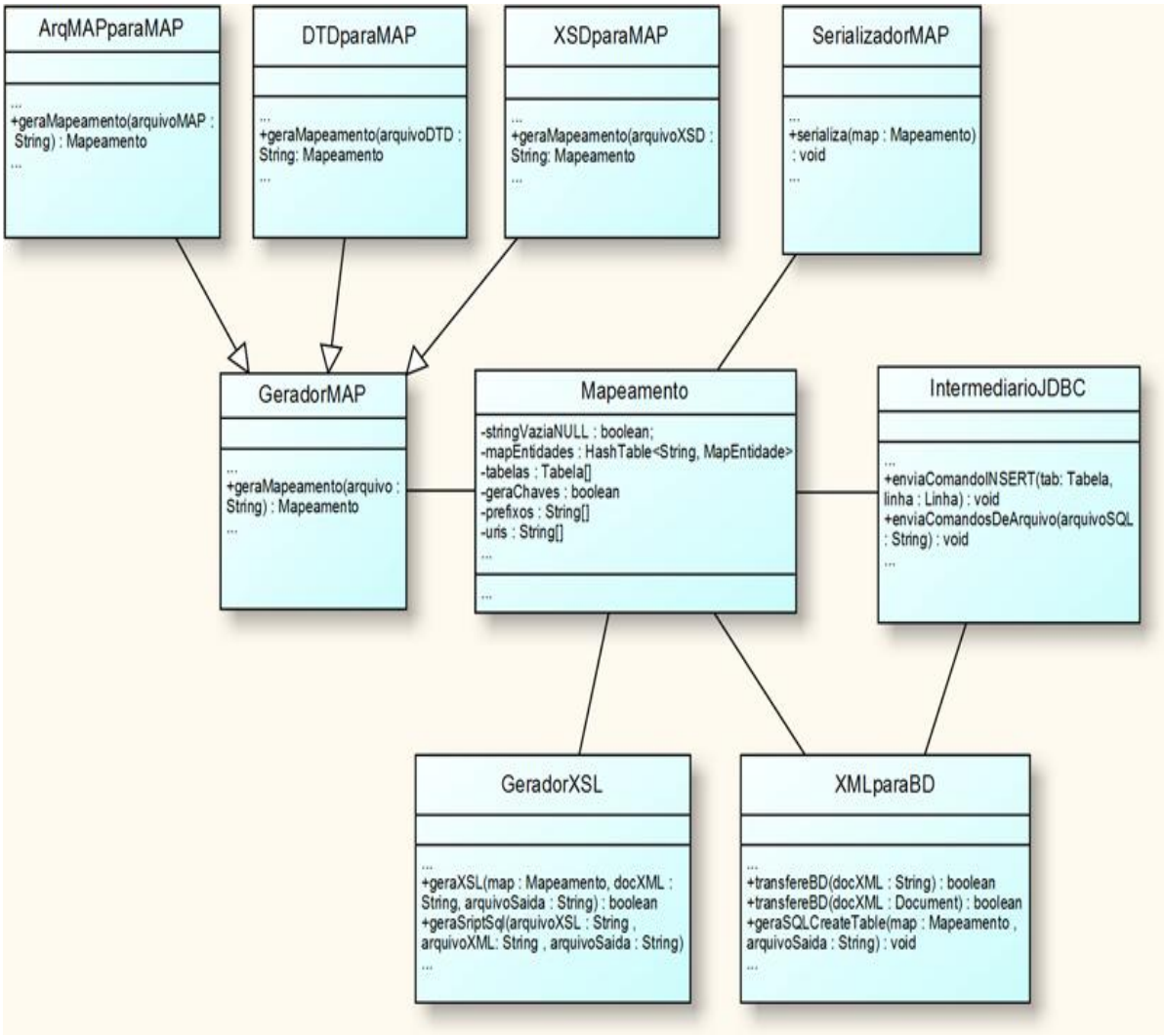

Figura 3: diagrama de classes

A classes ArqMAParaMap, DTDparaMAP e XSDparaMAP estendem a classe GeradorMAP e oferecem entre outras funções, a função geraMapeamento(String arquivo), que retorna um objeto da classe Mapeamento.

A classe SerializadorMAP oferece a função serializa(Mapeamento map, String arquivoSaida), que recebe um objeto Mapeamento e o serializa em forma de um arquivo map.

A classe GeradorXSL oferece a função *geraXSL(Mapeamento map, String saída)* que cria o arquivo de saída que contém comandos XSLT e a função estática *geraScriptSql(String arquivoXSL, String aqruivoXML, String arquivoSaida)* que

transforma o documento XML recebido usando os comandos XSLT contidos no arquivo XSL em um script SQL com comandos INSERT.

A classe IntermediarioJDBC oferece funções de acesso ao BD como por exemplo *enviaComandoINSERT(Tabela tab, Linha lin)*, *enviaComandosDeArquivo(String arquivoSQL)*, etc.

A classe XMLparaBD fornece entre outras funções, *transfereBD(String docXML)* e *transfereBD(Document docXML)*, que recebem como parâmetro o documento XML e retornam um valor booleano para sucesso ou não. Esta classe oferece ainda a função *getComandosCreateTable(Mapeamento map, String arquivoSaida)*, que recebe um objeto Mapeamento e gera um script SQL com comandos CREATE TABLE.

Foi criada uma classe XmlBDTool que é a ferramenta propriamente dita, que recebe comandos pela linha de comandos e executa uma das funcionalidades do sistema. Por exemplo, para fazer a carga do BD a partir de um documento XML, o comando é:

*java XmlBD*Tool -transf <arquivo_map> <doc_xml> <arquivo_conf>

Note-se que o nome do driver JDBC e a url de conexão com o BD, incluindo nome de usuário e senha, se existir, são especificados no arquivo de configuração em formato texto. Mais comandos oferecidos pela ferramenta, bem como o formato do arquivo de configuração serão mostrados no apêndice deste trabalho.

A seguir são apresentados alguns detalhes de implementação referentes às principais funcionalidades do sistema.

4.3.1 Geração do objeto Mapeamento

O Objeto Mapeamento pode ser encarado como a versão compilada de um arquivo de mapeamento. Ele contém informações referentes ao documento XML e a visão do BD, bem como metadados como o tipo de dado das colunas do BD, ou seja, todas as informações necessárias para a carga do BD.

A ferramenta tem a capacidade de gerar esse objeto de forma automática a partir de um arquivo de mapeamento estruturado de acordo com a linguagem de mapeamento, ou ainda um esquema XML (DTD ou XML Schema).

A função *geraMapeamento(String arquivo)* da superclasse GeradorMAP é implementada por cada uma das subclasses e retorna um objeto Mapeamento recebendo

como parâmetro o arquivo que pode ser um arquivo .map, um DTD ou ainda um arquivo XSD. No caso de um arquivo de mapeamento (que é um arquivo XML), o mesmo é analisado por um parser SAX que traduz a informação lida e o objeto Mapeamento é criado e preenchido durante o processo. O SAX (*Simple API for XML)* é uma API que provê acesso sequencial baseado em eventos, ao conteúdo de um documento XML. Os eventos SAX incluem nodos textos, nodos elementos, instruções de processamento e comentários. Tais eventos são disparados quando os nodos são encontrados durante o processo de parsing e novamente no final dos mesmos. A aplicação SAX implementa a interface ContentHandler para poder receber notificações de novos eventos. A interface oferece métodos como *startDocument(), endDocument(), startElement(...), endElement(...),* etc., que, quando implementados pela aplicação, indicam quais ações tomar quando for encontrado no documento XML determinado elemento, ou atributo, etc. A classe ArqMAPparaMAP do pacote 'Geradores' implementa as funções oferecidas pela interface ContentHandler.

Os arquivos DTD são analisados por um parser de código livre desenvolvido por Mark Wutka (2002). O parser retorna um objeto DTD similar a uma árvore DOM que, então, é processada.

Já os XML Schema são analisados por um parser DOM, visto que um arquivo XSD é também um arquivo XML. O DOM (*Document Object Model)* é uma especificação da W3C, independente de plataforma e linguagem, que permite alterar e editar dinamicamente a estrutura e conteúdo de um documento XML. O documento XML é carregado na memória em ma estrutura de árvore e então é oferecida uma API que permite percorrer e manipular tal árvore.

Existem diferentes implementações do SAX e DOM, de diferentes desenvolvedores, com características diferentes. Para contornar este problema e manter-se independente da implementação escolhida, foi usada a API JAXP *(Java API for XML Processing).* A JAXP faz uso dos padrões definidos pelo SAX e DOM e permite que a aplicação funcione corretamente independente da implementação escolhida do parser. Para isso, a API oferece interfaces genéricas chamadas *SaxParserFactory* e *DocumentBuilderFactory.* As duas funções a seguir mostram como fazer o uso dessas interfaces para se obter um parser SAX e para instanciar na memória uma arvore DOM a partir de um documento XML, sem se preocupar com qual a implementação SAX e DOM será usada:

```
static XMLReader  getSAXParser()
{
          SAXParser parser = null;
          XMLReader reader = null;
          try{
                  //Instancia o parser e seta opcoes
                  SAXParserFactory factory = SAXParserFactory.newInstance();
                  factory.setValidating(true);
                  factory.setNamespaceAware(true);
                  parser = factory.newSAXParser();
                  reader = parser.getXMLReader();
          } catch (Exception e) {
                  e. printStackTrace();
         }

          return reader;
   }
```

Figura 4: função que instancia um parser SAX independente da implementação

```
static Document abreDocumento (String arquivoXML) throws Exception
 {
        Document doc = null;
       try{
                  DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();

                // Seta a validaçao e o uso de namespaces
                fac.setValidating(true);
                fac.setNamespaceAware(false);

              // Obtem o DocumentBuilder

                DocumentBuilder builder = fac.newDocumentBuilder();
                doc = builder.parse(new InputSource(getFileURL(arquivoXML)));

              } catch (Exception e) {
              e.printStackTrace();
             }

        return doc;
 }
```

Figura 5: função que cria uma árvore DOM a partir de um documento XML, independente da implementação DOM

4.3.2 Geração dos comandos CREATE TABLE

Um script SQL pode ser gerado automaticamente pela ferramenta com comandos CREATE TABLE, aproveitando-se de facilidades do uso do JDBC. A interface DatabaseMetaData, que é implementada pelos desenvolvedores dos drivers JDBC fornece várias informações específicas de cada SGDB e informações gerais do BD.

O script gerado pode ser depois editado manualmente pelo usuário ou usado para gerar as tabelas em outros BDs, para eventuais futuras cargas.

4.3.3 Carga do BD a partir de documentos XML

O documento XML que contém os dados é analisado por um parser DOM que gera uma estrutura em forma de árvore com os nodos do documento. Nodos em uma árvore DOM podem ser elementos, comentários, instruções de processamento, textos, atributos, entidades, referências a entidades, etc. No entanto, muitos desses nodos não devem ser processados e por isso devem ser ignorados. Outros devem ser modificados ou expandidos. O algoritmo que percorre a árvore, a modifica conforme necessário e comporta como se a árvore fosse composta apenas por nodos lógicos. Tais nodos podem ser nodos elemento, nodos atributo e nodos texto totalmente normalizados.

Nodos de comentários e instruções de processamento são descartados, nodos textos e CDATA adjacentes são fundidos, e nodos de referência a entidades são substituídos pelos seus filhos. Por exemplo, consideremos a árvore DOM da Figura 6:

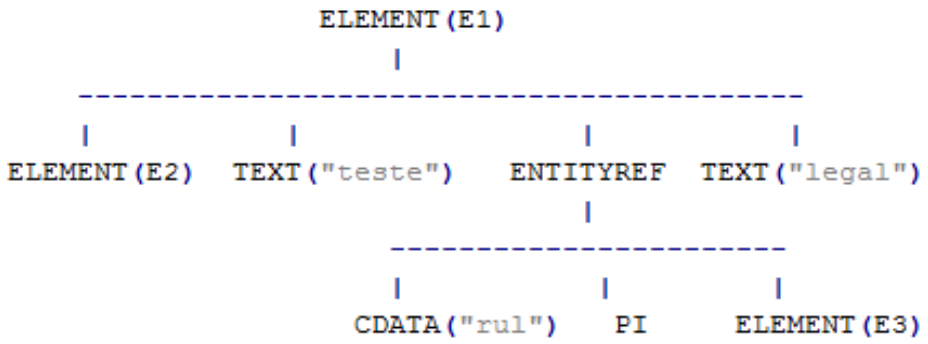

Figura 6: Árvore DOM original antes do processamento

O primeiro nodo lógico filho do nodo E1 é o nodo elemento E2. O nodo irmão (sibling node) do elemento E2 é o elemento texto “testerul”, que foi a fusão do nodo texto “teste” e do nodo CDATA “rul”. O elemento lógico irmão do nodo texto “testerul” é o elemento E3. O elemento texto “legal” fica inalterado, pois após o processamento da ENTITYREF e da fusão dos nodos texto e CDATA adjacentes, passou a ser nodo irmão do elemento E3. Após o processamento descrito, a árvore DOM resultante pode ser vista na Figura 7.

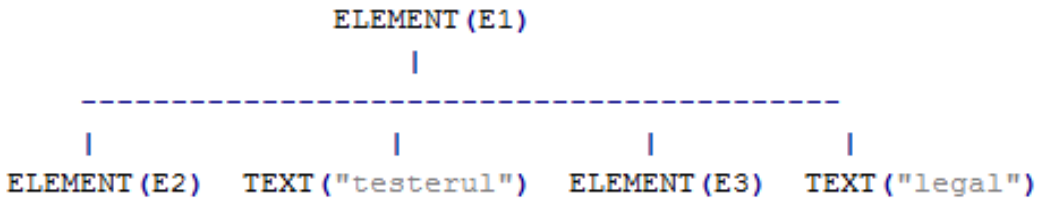

Figura 7: Árvore DOM resultante após o processamento

A árvore DOM é percorrida recursivamente numa combinação entre busca por largura e por profundidade, de modo a preservar a integridade referencial do BD.

Quando é encontrado um elemento que será mapeado para uma tabela (elemento visto como entidade), é criado um buffer na memória para uma "linha" daquela tabela. Em seguida, todos os nodos filhos do elemento são processados. Se um filho é mapeado para uma coluna (elemento visto como propriedade ou atributo), o valor do mesmo é armazenado no buffer. Se um filho é mapeado para uma tabela (elemento visto como entidade), o relacionamento entre as tabelas é verificado. Caso o relacionamento for do tipo um para um ou um para N e se a chave estrangeira deve ser armazenada na tabela atual (correspondente ao elemento pai), o nodo filho encontrado é processado imediatamente e a chave estrangeira é armazenada no buffer. Caso contrário, ou seja, a chave única pertence à tabela do pai, o nodo filho é salvo em uma pilha para posterior processamento. Isto se deve ao fato de que o mesmo só pode ser processado e inserido no BD depois que o elemento pai já tiver sido inserido no banco.

O relacionamento do tipo "N-N" existe quando é encontrado um elemento filho que foi mapeado através de uma tabela associativa ou uma referência ao mesmo através de um IDREF. Outra situação é quando o elemento filho encontrado não foi mapeado para nenhuma tabela. Neste caso, verificam-se os filhos deste elemento na procura de algum elemento mapeado através de tabela associativa ou uma referência ao mesmo. Caso seja encontrado, é criado um buffer para cada "linha" da tabela associativa a ser inserida. Em caso de uso de referências a elementos, a função getElementById (String elementid) oferecida pela interface Document é usada para localizar o elemento desejado.

Depois de processados todos os nodos filhos, a linha corrente é então inserida no BD através do JDBC ou é criado um arquivo script SQL com os comandos INSERT gerados.

Por fim, os nodos filhos que foram armazenados na pilha são processados e recebem o valor da chave única do elemento pai que acabou de ser processado. No caso

da existência de algum relacionamento N-N, após os elementos envolvidos terem sido inseridos no BD, o buffer com as linhas da tabela associativa é descarregado para o BD, recebendo antes as chaves das duas tabelas envolvidas, conforme especificado no mapeamento.

Durante o processamento do relacionamento entre as tabelas, caso tenha sido especificado no arquivo de mapeamento que a chave de uma tabela deve ser gerada automaticamente, o objeto da classe GeradorChaves é invocado e o mesmo devolve uma chave. O algoritmo usado na geração de chaves únicas é um dos clássicos algoritmos, que consegue gerar um inteiro de 32 bits combinando os bits de dois outros inteiros (highkey e lowkey). Os 24 bits mais significativos são formados pelo highkey e os 8 bits menos significativos pelo lowkey. O highkey pode ser armazenado em ma tabela no BD ou localmente no sistema de arquivos e inicialmente tem valor 1. O lowkey é inicializado com 0 e é incrementado de 1 cada vez que o método gerarChaveUnica() for chamado. Quando ele atingir o valor de 255, o highkey é incrementado de uma unidade. Isso significa que 256 chaves únicas podem ser geradas com um valor de highkey. A vantagem de tê-lo armazenado no BD é que seu valor estará sempre atualizado e nunca uma mesma chave será repetida. Com um único acesso ao BD, 256 chaves diferentes podem ser geradas.

Documentos XML são internacionais e como consequência, pode-se neles encontrar uma variedade de formatos de data, hora e timestamp. Por default, o sistema usa os dados do Locale da máquina usada, para ler as datas, horas e timestamps no documento XML, usando as classes DateFormat e SimpleDateFormat do Java. Porém, é possível indicar ao sistema outros formatos para tratar esses dados.

4.3.4 Geração do arquivo XSL com comandos XSLT

O objeto Mapeamento, que contém toda a informação sobre o mapeamento de cada elemento do documento XML, quando passado a um objeto da classe GeradorXSL, permite que comandos XSLT sejam escritos num arquivo XSL, usando comandos XPath para localizar os elementos no documento XML. A geração dos comandos XSLT é feita principalmente através da combinação do uso de elementos xsl:template. O elemento <xsl:template> contém regras que devem ser aplicadas quando um nodo especificado é encontrado. O atributo ‘match’ é usado para associar um template a um elemento XML.

O elemento <xsl:apply-templates> aplica um template ao elemento atual, ou aos filhos do mesmo. Pode ser utilizado o atributo ‘select’, que restringe o processamento a apenas os elementos filhos que correspondam ao seu valor. Este atributo também é usado para especificar a ordem em que os nodos filhos devem ser processados, o que é importante, pois existe uma ordem específica para a execução de comandos INSERT, de modo a garantir a integridade referencial do BD.

O elemento <xsl:value-of> é usado para extrair o valor de um elemento ou atributo XML e adicionar o mesmo à saída do transformador XSLT. O atributo ‘select’ novamente pode ser usado para indicar atributos e elementos específicos.

Considere o documento XML a seguir, que contém dados de funcionários organizados por departamento:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<departamentos>
	<departamento id ="1">
		<nome>Comercial</nome>
		<funcionario id="345">
			<nome>Alberto Martins</nome>
			< sexo >M</ sexo >
		</funcionario>
		<funcionario id="56">
			<nome>Joao Pereira</nome>
			< sexo >M</ sexo >
		</funcionario>
	</departamento>
	<departamento id ="2">
		<nome>Financeiro</nome>
		<funcionario id="788">
			 <nome>Julia Souza</nome>
			< sexo >F</ sexo >
		</funcionario>
	</departamento>
	<departamento id ="4">
		<nome>Desenvolvimento</nome>
		<funcionario id="567">
			 <nome>Manuel Monteiro</nome>
			<sexo>M</ sexo >
		</funcionario>
	</departamento>
</departamentos>
```

O documento XSL correspondente gerado é o seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:output method="text" name="sql" omit-xml-declatration="yes"/>
```

```
<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="departamento">

    INSERT INTO TDEPARTAMENTO (ID, CNOME)
    VALUES (<xsl:value-of select="@id"/>, '<xsl:value-of select="nome"/>');

    <xsl:apply-templates select="funcionario"/>
</xsl:template>

<xsl:template match="funcionario">

    INSERT INTO TFUNCIONARIO (ID, CNOME, SEXO, DID)
    VALUES (<xsl:value-of select="@id"/>, '<xsl:value-of select="nome"/>',
        '<xsl:value-of select="sexo"/>', <xsl:value-of select="../@id"/>);
</xsl:template>

</xsl:stylesheet>
```

O primeiro template declarado especifica que quando for encontrado o elemento raiz do documento, os templates declarados na sequência devem ser aplicados.

O template seguinte especifica que cada vez que for encontrado um elemento ‘departamento’, o comando insert correspondente deve ser escrito e em seguida o elemento ‘xsl:apply-templates’ aplica o template a todos os seus elementos filhos de nome ‘funcionario’, que gera os comandos insert para cada um, conforme especificado no terceiro template. O elemento ‘xsl:value-of‘ é usado para extrair os valores dos elementos simples e dos atributos. O atributo ‘select’ indica algum comando XPath. No caso de um atributo, como por exemplo, o id, o valor é obtido através do comando “@id”. No template funcionário, como o id do departamento deve ser inserido também, é usado o comando “../@id” que obtém o id do elemento imediatamente ascendente ao elemento funcionário, no caso, o elemento departamento correspondente.

Este script, quando aplicado por um transformador XSLT sobre o documento XML original, gera como saída um script SQL com os comandos INSERT:

```
INSERT INTO TDEPARTAMENTO (ID, CNOME)
VALUES (1, 'Comercial');

INSERT INTO TFUNCIONARIO (ID, CNOME, SEXO, DID)
VALUES (345, 'Alberto Martins', 'M', 1);

INSERT INTO TFUNCIONARIO (ID, CNOME, SEXO, DID)
VALUES (56, 'Joao Pereira', 'M', 1);
```

```
INSERT INTO TDEPARTAMENTO (ID, CNOME)
VALUES (2, 'Financeiro');

INSERT INTO TFUNCIONARIO (ID, CNOME, SEXO, DID)
VALUES (788, 'Julia Souza', 'F', 2);

INSERT INTO TDEPARTAMENTO (ID, CNOME)
VALUES (4, 'Desenvolvimento');

INSERT INTO TFUNCIONARIO (ID, CNOME, SEXO, DID)
VALUES (567, 'Manuel Monteiro', 'M', 4);
```

Uma vantagem de gerar um documento XSL com os comandos XSLT é que uma vez gerado, ele pode ser aplicado a toda uma família de documentos XML e não apenas a aquele documento específico. Por exemplo, no exemplo dado, caso houvesse mais documentos naquele formato mas com diferentes dados, o script XSLT gerado poderia ser usado para gerar os comandos INSERT do SQL.

Outra vantagem é que o script pode ser facilmente editado por um usuário, para, por exemplo, alterar o nome das tabelas e/ou colunas, ou ainda acrescentar ou remover alguma informação através da inserção de outros comandos XSLT.

4.3.5 Acesso ao BD

A ferramenta permite que todo o acesso ao BD seja feito através do JDBC (*Java Database Connectivity*) que é um conjunto de classes e interfaces (API) escritas em Java que fazem o envio de instruções SQL para qualquer BD relacional. O acesso ao BD através da API JDBC garante a independência da aplicação em relação ao SGBD destino. Basta que exista um driver JDBC para este SGBD, disponibilizados por praticamente todos os desenvolvedores, para que o acesso ao banco seja feito, sem nenhuma mudança no código da aplicação e sem conhecer previamente o SGBD que será utilizado. O driver não é mais do que uma implementação específica dos comandos genéricos do JDBC para um determinado SGBD.

Além de permitir enviar comandos SQL para o BD, o JDBC oferece facilidades na obtenção de informações referentes ao BD em questão, através da interface DatabaseMetaData.

Como opção, o usuário pode utilizar os scripts SQL com comandos CREATE TABLE e INSERT gerados pela ferramenta, de forma direta sem o uso do JDBC. Caso o JDBC seja utilizado, o usuário tem a opção de especificar se o commit deve ser feito

imediatamente após cada operação de insert ou apenas no final, após todos os dados já estiverem inseridos. Além disso, é possível fazer uma validação dos nomes das tabelas, colunas e tipos, antes da inserção dos dados, evitando erros durante a execução dos comandos INSERT.

4.4 Validação

Por se tratar de uma ferramenta de carga de BDs relacionais para fontes de dados XML, para ser considerada uma ferramenta válida e usável no dia a dia, dois pontos importantes devem ser considerados: a integridade referencial e o suporte aos diferentes tipos de relacionamento entre entidades.

Em um BD relacional, a integridade referencial garante a não corrupção dos dados, de modo a não existir um registro "filho" sem um registro "pai". Quando um registro aponta para outro e depende deste, são necessárias regras para que o registro "pai" não possa ser excluído se ele tiver "filhos" (as suas dependências). O relacionamento é feito através das chaves estrangeiras das tabelas, avaliadas antes da execução dos comandos *delete*, *insert* ou *update*. Em outras palavras, quando uma tabela possui uma coluna com uma chave estrangeira, este campo deve apontar para uma chave única em outra tabela ou um valor nulo. Como a ferramenta apenas permite a inserção de dados, deve ser garantido que quando uma “linha” de uma tabela que possui uma chave estrangeira que aponte para outra tabela for ser inserida no banco, a linha correspondente da tabela que contém a chave primária já tenha sido inserida anteriormente. Conforme explicado na seção de implementação, o algoritmo implementado respeita esta regra e mantém num buffer de memória o registro a ser inserido caso o mesmo possua uma chave estrangeira, até que o registro com a chave única correspondente tenha sido inserido.

Em relação aos diferentes tipos de relacionamento, a ferramenta oferece suporte aos tipos um para um, um para N e N para N.

No estudo de caso da seção seguinte, será apresentado um exemplo completo juntamente com os resultados obtidos, numa forma de validar a ferramenta.

4.4.1 Estudo de caso

Como estudo de caso, foi escolhido um cenário simples e fictício de uma universidade, cujo BD armazena informações sobre os alunos, professores, turmas,

disciplinas e as relações entre os mesmos, conforme mostra o diagrama de entidade-relacionamento da Figura 8.

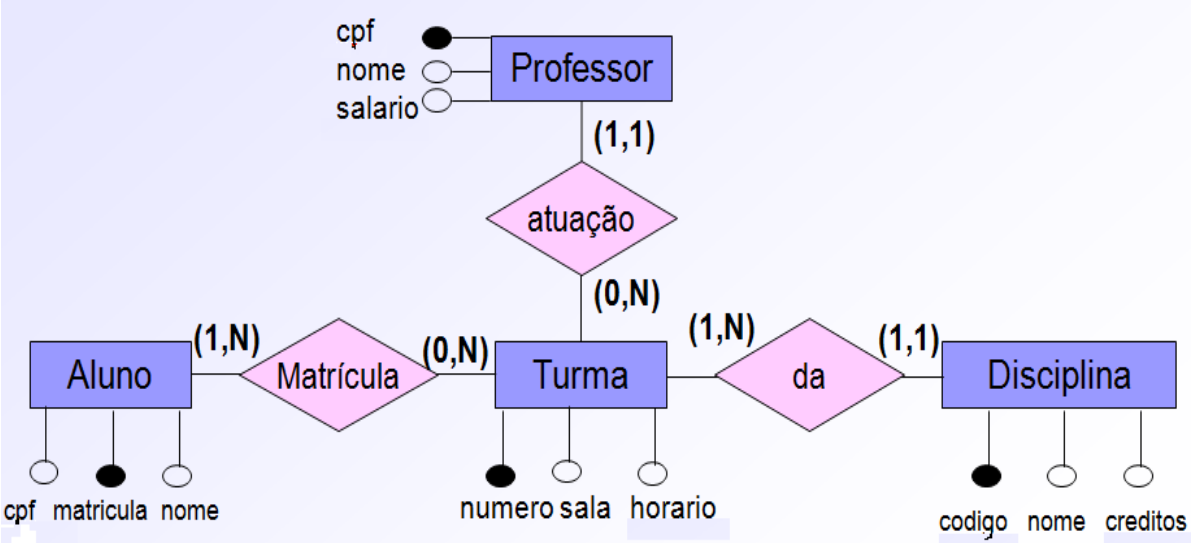

Figura 8: modelo entidade-relacionamento do estudo de caso

Conforme mostrado na figura acima, as entidades envolvidas são: Aluno, Professor, Disciplina e Turma. Entre a entidade Aluno e Turma há um relacionamento de muitos para muitos. Entre as entidades Disciplina e Turma existe um relacionamento de um para muitos, assim como entre as entidades Professor e Turma. No exemplo apresentado, foi utilizado o SGBD MySQL para realizar o teste, bem como o driver JDBC disponível para o mesmo. Foram considerados dois cenários. No primeiro, um documento XML contendo dados foi usado para fazer a carga do BD e foi verificado o estado final das tabelas no banco após a carga. No segundo, com as tabelas já criadas no BD, foi feita a exportação das mesmas para documentos XML através do utilitário *mysqldump* e em seguida foi feita a carga do BD a partir dos documentos gerados para comparar o resultado com as tabelas originais.

4.4.1.1 Cenário 1

O documento XML que contém os dados que serão usados na carga do BD é o seguinte:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Universidade SYSTEM "universidade.dtd">

<Universidade>

   <Aluno matricula="12516LV">
    <cpf>28462887-3</cpf>
```

```
    <nome>Miguel Santos</nome>
  </Aluno>
  <Aluno matricula="B23245">
    <cpf>09334734-5</cpf>
    <nome>Maria Cristina Gomes</nome>
  </Aluno>
  <Aluno matricula="3P4235">
    <cpf>98348782-7</cpf>
    <nome>João Delgado</nome>
  </Aluno>
  <Aluno matricula="265365">
    <cpf>28462887-3</cpf>
    <nome>Miguel Santos</nome>
  </Aluno>
  <Aluno matricula="125192">
    <cpf>2842342-3</cpf>
    <nome>Joao Pires Silva</nome>
  </Aluno>


  <Turma numero="98" sala="ctc210">
    <horario>08:10</horario>
    <estudantes>
      <aluno-ref matr="3P4235"/>
      <aluno-ref matr="265365"/>
      <aluno-ref matr="125192"/>
    </estudantes>
    <Professor cpf="8738787-7">
      <nome>Franscisco Pires</nome>
      <salario>5390</salario>
    </Professor>
    <Disciplina codigo="IRF098">
      <nome>Anatomia I</nome>
      <creditos>5</creditos>
    </Disciplina>
  </Turma>


  <Turma numero="67" sala="ctc112">
    <horario>10:20</horario>
    <estudantes>
      <aluno-ref matr="12516LV"/>
      <aluno-ref matr="B23245"/>
      <aluno-ref matr="125192"/>
      <aluno-ref matr="3P4235"/>
    </estudantes>
    <Professor cpf="232344-9">
      <nome>Manuel Monteiro</nome>
      <salario>4815</salario>
    </Professor>
    <Disciplina codigo="POL87">
      <nome>Nataçao II</nome>
      <creditos>3</creditos>
    </Disciplina>
  </Turma>

</Universidade>
```

Foram criadas no BD as tabelas TAluno, TProfessor, TDisciplina, TTurma e a tabela TMatricula que é a tabela associativa entre TAluno e TTurma.

O arquivo de mapeamento XML-Relacional é apresentado a seguir:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE XMLparaBD SYSTEM "C:\Users\Edwaldo\tcc\XMLparaBD.dtd">
<XMLparaBD Versao="1.0">
  <MapXMLRel>

    <!-- Ignora o elemento Universidade e mapeia Aluno e Turma -->
    <IgnoraElementoRaiz>
      <Elemento Nome="Universidade"/>
      <FilhoRaiz>
        <Elemento Nome="Aluno"/>
      </FilhoRaiz>
      <FilhoRaiz>
        <Elemento Nome="Turma"/>
      </FilhoRaiz>
    </IgnoraElementoRaiz>

    <!-- Mapeamento de Aluno -->
    <Entidade>
      <Elemento Nome="Aluno"/>
      <Tabela Nome="TAluno"/>
      <Propriedade>
        <Atributo Nome="matricula"/>
        <Coluna Nome="matricula"/>
      </Propriedade>
      <Propriedade>
        <Elemento Nome="nome"/>
        <Coluna Nome="nome"/>
      </Propriedade>
      <Propriedade>
        <Elemento Nome="cpf"/>
        <Coluna Nome="cpf"/>
      </Propriedade>
    </Entidade>

    <!-- Mapeamento de Turma -->
    <Entidade>
      <Elemento Nome="Turma"/>
      <Tabela Nome="TTurma"/>
      <Propriedade>
        <Atributo Nome="numero"/>
        <Coluna Nome="numero"/>
      </Propriedade>
      <Propriedade>
        <Atributo Nome="sala"/>
        <Coluna Nome="sala"/>
      </Propriedade>
      <Propriedade>
        <Elemento Nome="horario"/>
        <Coluna Nome="horario"/>
      </Propriedade>
```

```
<!-- Mapeamento do relacionamento N-N entre Aluno e Turma -->
    <EntidadeRelacionada tipoRelacionamento="N-N">
      <Elemento nome="Aluno"/>
      <TabelaAssociativa nome="TMatricula" elemento="estudantes/aluno-ref" idref="matr">
        <Coluna nome="matr_aluno" valor="TAluno.matricula"/>
        <Coluna nome="num_turma" valor="TTurma.numero"/>
      </TabelaAssociativa>
    </EntidadeRelacionada>

    <!-- Mapeamento do relacionamento 1-N entre Professor e Turma-->
    <EntidadeRelacionada tipoRelacionamento="Direto">
      <Elemento nome="Professor"/>
      <ChavesJuncao ChaveEntidadePai="Estrangeira">
        <ChaveUnica gerarAutomaticamente="N">
          <Coluna nome ="cpf"/>
        </ChaveUnica>
        <ChaveEstrangeira>
          <Coluna nome ="cpf_professor"/>
        </ChaveEstrangeira>
      </ChavesJuncao>
    </EntidadeRelacionada>

    <!-- Mapeamento do relacionamento 1-N entr Disciplina e Turma-->
    <EntidadeRelacionada tipoRelacionamento="Direto">
      <Elemento nome="Disciplina"/>
      <ChavesJuncao ChaveEntidadePai="Estrangeira">
        <ChaveUnica gerarAutomaticamente="N">
          <Coluna nome ="codigo"/>
        </ChaveUnica>
        <ChaveEstrangeira>
          <Coluna nome ="cod_disciplina"/>
        </ChaveEstrangeira>
      </ChavesJuncao>
    </EntidadeRelacionada>

  </Entidade>

  <!-- Mapeamento de Professor -->
  <Entidade>
    <Elemento Nome="Professor"/>
    <Tabela Nome="TProfessor"/>
    <Propriedade>
      <Atributo Nome="cpf"/>
      <Coluna Nome="cpf"/>
    </Propriedade>
    <Propriedade>
      <Elemento Nome="nome"/>
      <Coluna Nome="nome"/>
    </Propriedade>
    <Propriedade>
      <Elemento Nome="salario"/>
      <Coluna Nome="salario"/>
    </Propriedade>
  </Entidade>

  <!-- Mapeamento de Disciplina -->
  <Entidade>
    <Elemento Nome="Disciplina"/>
```

```
    <Tabela Nome="TDisciplina"/>
    <Propriedade>
      <Atributo Nome="codigo"/>
      <Coluna Nome="codigo"/>
    </Propriedade>
    <Propriedade>
      <Elemento Nome="nome"/>
      <Coluna Nome="nome"/>
    </Propriedade>
    <Propriedade>
      <Elemento Nome="creditos"/>
      <Coluna Nome="creditos"/>
    </Propriedade>
   </Entidade>

  </MapXMLRel>
</XMLparaBD>
```

As tabelas inicialmente estavam vazias, conforme ilustrado no prompt mysql da Figura 9.

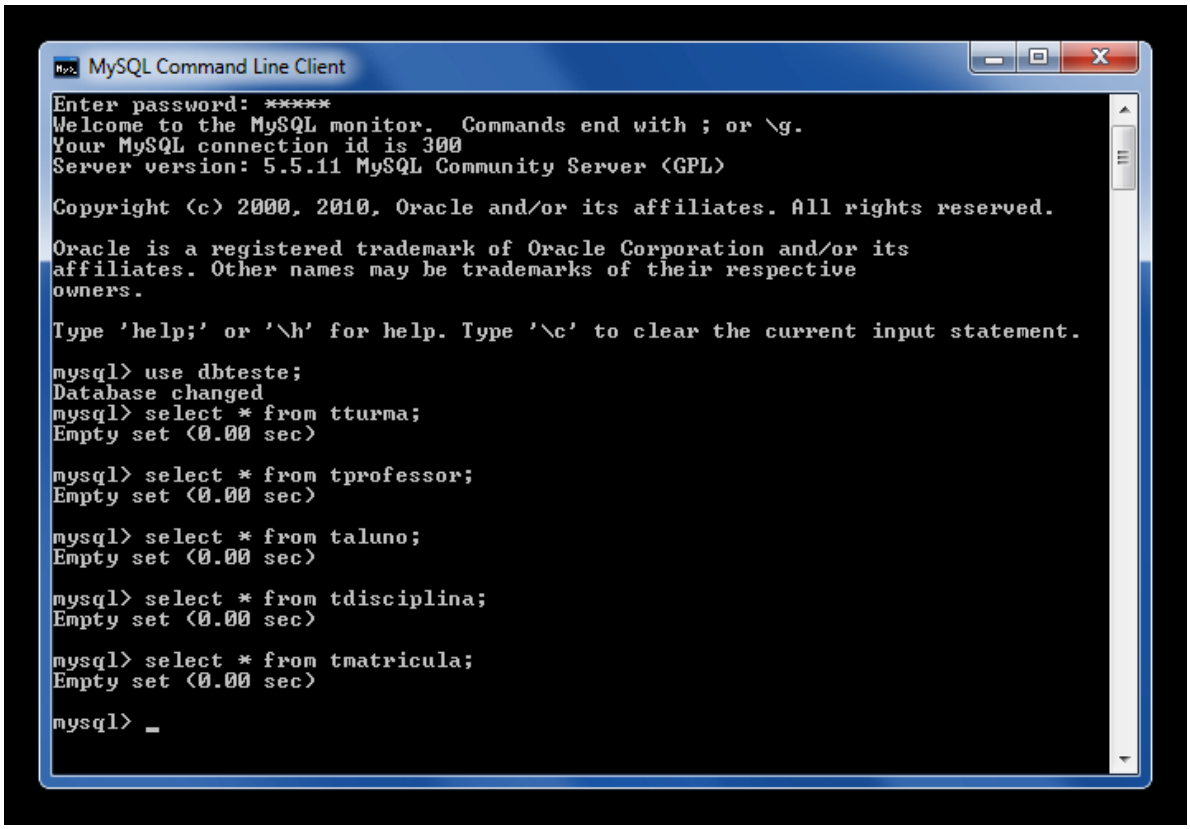

Figura 9: Tabelas criadas no BD antes da execução da ferramenta

Em seguida foi executada a ferramenta conforme mostra a Figura 10.

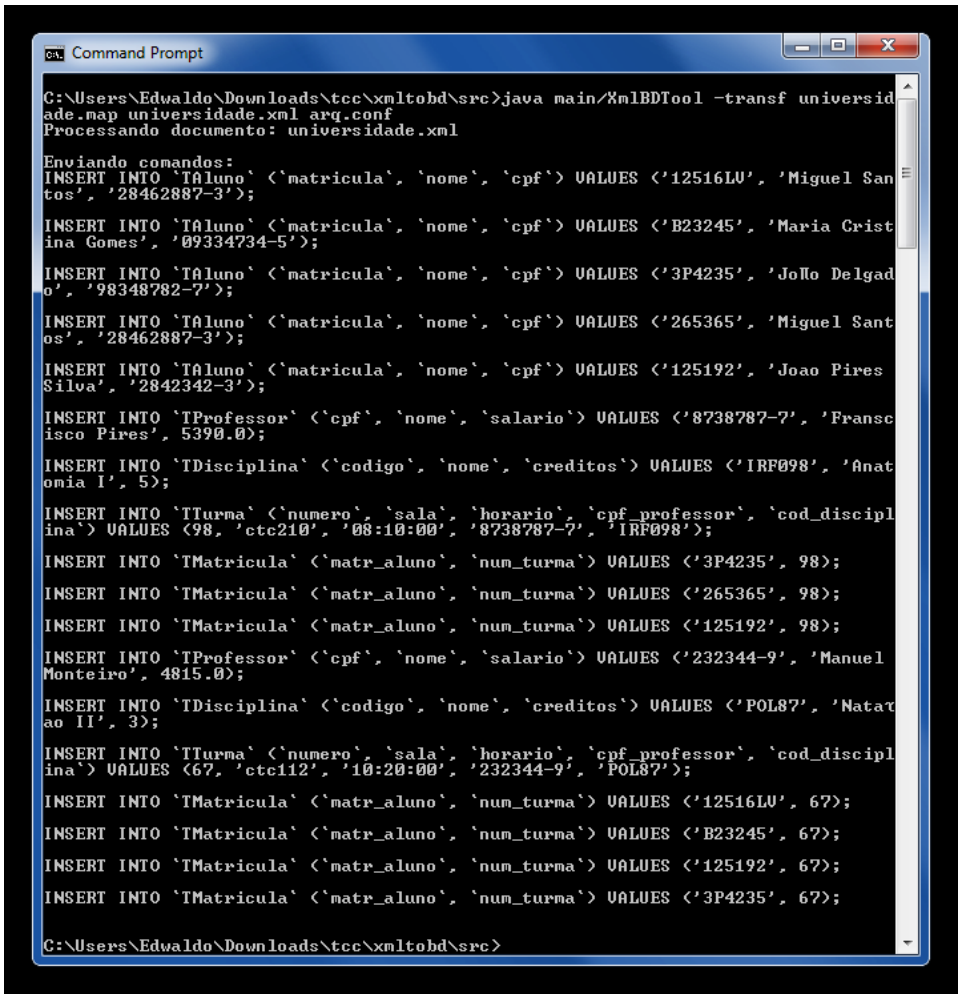

Figura 10: Execução da ferramenta passando o documento XML e o mapeamento

Os comandos INSERT do SQL que foram enviados pelo JDBC ao BD foram:

INSERT INTO `TAluno` (`matricula`, `nome`, `cpf`) VALUES ('12516LV', 'Miguel Santos', '28462887-3');

INSERT INTO `TAluno` (`matricula`, `nome`, `cpf`) VALUES ('B23245', 'Maria Cristina Gomes', '09334734-5');

INSERT INTO `TAluno` (`matricula`, `nome`, `cpf`) VALUES ('3P4235', 'João Delgado', '98348782-7');

INSERT INTO `TAluno` (`matricula`, `nome`, `cpf`) VALUES ('265365', 'Miguel Santos', '28462887-3');

INSERT INTO `TAluno` (`matricula`, `nome`, `cpf`) VALUES ('125192', 'Joao Pires Silva', '2842342-3');

```
INSERT INTO `TProfessor` (`cpf`, `nome`, `salario`) VALUES ('8738787-7', 'Franscisco Pires', 5390.0);

INSERT INTO `TDisciplina` (`codigo`, `nome`, `creditos`) VALUES ('IRF098', 'Anatomia I', 5);

INSERT INTO `TTurma` (`numero`, `sala`, `horario`, `cpf_professor`, `cod_disciplina`) VALUES (98, 'ctc210', '08:10:00', '8738787-7', 'IRF098');

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('3P4235', 98);

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('265365', 98);

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('125192', 98);

INSERT INTO `TProfessor` (`cpf`, `nome`, `salario`) VALUES ('232344-9', 'Manuel Monteiro', 4815.0);

INSERT INTO `TDisciplina` (`codigo`, `nome`, `creditos`) VALUES ('POL87', 'Nataçao II', 3);

INSERT INTO `TTurma` (`numero`, `sala`, `horario`, `cpf_professor`, `cod_disciplina`) VALUES (67, 'ctc112', '10:20:00', '232344-9', 'POL87');

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('12516LV', 67);

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('B23245', 67);

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('125192', 67);

INSERT INTO `TMatricula` (`matr_aluno`, `num_turma`) VALUES ('3P4235', 67);
```

Por fim, foi verificado o estado das tabelas no BD e o resultado foi o esperado, conforme mostra a Figura 11.

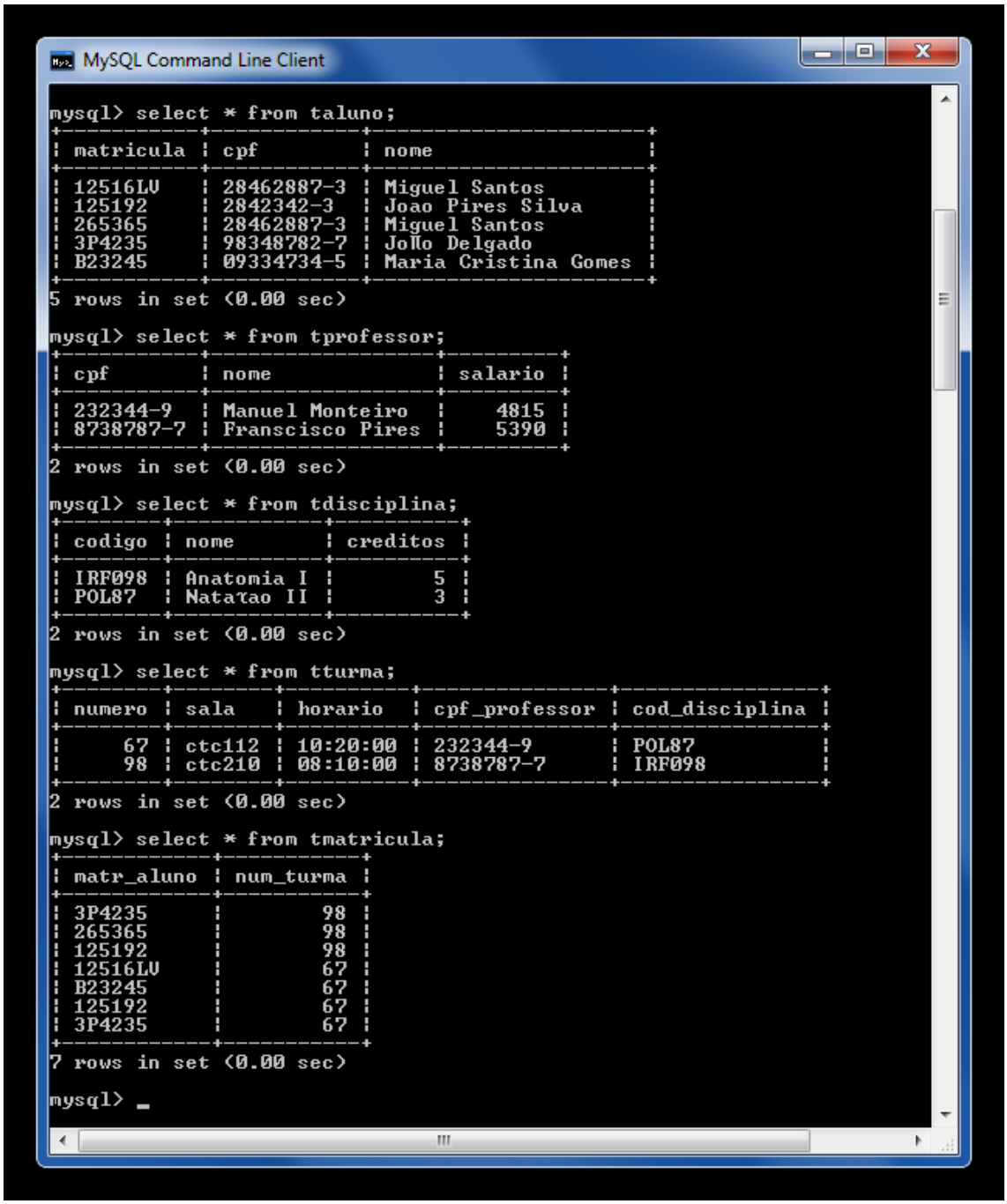

Figura 11: Estado final das tabelas do BD após a execução da ferramenta

4.4.1.2 Cenário 2

O *mysqldump* é um utilitário do MYSQL que permite fazer backups de BDs, gerar scripts SQL e documentos XML a partir dos BDs existentes e fazer a restauração dos mesmos.

A sintaxe de uso é:

mysqldump [options] db_name [tb1_name ...]

No caso específico da exportação de tabelas para documentos XML, a sintaxe é a seguinte:

mysqldump --xml db_name [tb1_name ...] > docXML

O mysqldump gera documentos XML estruturados da seguinte forma:

```
<Tabela>
        ...
        <Linha>
                ...
                <Coluna>
                </Coluna>
                ...
        </Linha>
        ...
</Tabela>
```

Este formato de documento XML é muito utilizado por outros SGBDs e suas extensões, para fazer a exportação e importação de documentos XML, pois se trata de um formato simples e de mapeamento trivial. O mapeamento é simples: os elementos 'Linha' são mapeados como entidades e correspondem a tuplas de alguma tabela, e os elementos 'Coluna' são mapeados como propriedades e correspondem a colunas da tabela correspondente ao elemento pai 'Linha'.

Os documentos XML gerados para as tabelas do BD são mostrados a seguir. Para efeito de simplificação foi omitido algumas informações irrelevantes para o mapeamento.

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <database name="dbteste">
        <table_structure name="taluno">
                ...
        </table_structure>

        <table_data name="taluno">
            <row>
                <matricula>12516LV</matricula>
                <nome>Miguel Santos</nome>
                <cpf>28462887-3</cpf>
            </row>
            <row>
                <matricula>125192</matricula>
                <nome>Joao Pires Silva</nome>
                <cpf>2842342-3</cpf>
            </row>
            <row>
                <matricula>265365</matricula>
                <nome>Miguel Santos</nome>
                <cpf>28462887-3</cpf>
```

```
            </row>
            <row>
                <matricula>3P4235</matricula>
                <nome>João Delgado</nome>
                <cpf>98348782-7</cpf>
            </row>
            <row>
                <matricula>B23245</matricula>
                <nome>Maria Cristina Gomes</nome>
                <cpf>09334734-5</cpf>
            </row>
        </table_data>
    </database>
</mysqldump>
```

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <database name="dbteste">

        <table_structure name="tdisciplina">
            ...
        </table_structure>

        <table_data name="tdisciplina">
            <row>
                <codigo>IRF098</codigo>
                <nome>Anatomia I</nome>
                <creditos>5</creditos>
            </row>
            <row>
                <codigo>POL87</codigo>
                <nome>Natação II</nome>
                <creditos>3</creditos>
            </row>
        </table_data>
    </database>
</mysqldump>
```

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <database name="dbteste">

        <table_structure name="tprofessor">
            ...
        </table_structure>

        <table_data name="tprofessor">
            <row>
                <cpf>232344-9</cpf>
                <nome>Manuel Monteiro</nome>
                <salario>4815</salario>
            </row>
            <row>
                <cpf>8738787-7</cpf>
                <nome>Franscisco Pires</nome>
                <salario>5390</salario>
            </row>
        </table_data>
    </database>
</mysqldump>
```

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <database name="dbteste">

        <table_structure name="tturma">
            ...
        </table_structure>

        <table_data name="tturma">
            <row>
                <numero>67</numero>
                <sala>ctc112</sala>
                <horario>10:20:00</horario>
                <cpf_professor>232344-9</cpf_professor>
                <cod_disciplina>POL87</cod_disciplina>
            </row>
            <row>
                <numero>98</numero>
                <sala>ctc210</sala>
                <horario>08:10:00</horario>
                <cpf_professor>8738787-7</cpf_professor>
                <cod_disciplina>IRF098</cod_disciplina>
            </row>
        </table_data>
    </database>
</mysqldump>


<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <database name="dbteste">

        <table_structure name="tmatricula">
            ...
        </table_structure>

        <table_data name="tmatricula">
            <row>
                <matr_aluno>3P4235</matr_aluno>
                <num_turma>98</num_turma>
            </row>
            <row>
                <matr_aluno>265365</matr_aluno>
                <num_turma>98</num_turma>
            </row>
            <row>
                <matr_aluno>125192</matr_aluno>
                <num_turma>98</num_turma>
            </row>
            <row>
                <matr_aluno>12516LV</matr_aluno>
                <num_turma>67</num_turma>
            </row>
            <row>
                <matr_aluno>B23245</matr_aluno>
                <num_turma>67</num_turma>
            </row>
            <row>
                <matr_aluno>125192</matr_aluno>
                <num_turma>67</num_turma>
            </row>
            <row>
                <matr_aluno>3P4235</matr_aluno>
                <num_turma>67</num_turma>
```

```
            </row>
        </table_data>
    </database>
</mysqldump>
```

O arquivo de mapeamento para a entidade Aluno é mostrado em seguida. O mapeamento das outras entidades é praticamente o mesmo, mudando apenas os nomes das tabelas e colunas. O elemento ‘row’ é mapeado como entidade e os seus elementos filhos como suas propriedades.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE XMLparaBD SYSTEM "C:\Users\Edwaldo\tcc\XMLparaBD.dtd">
<XMLparaBD Versao="1.0">
    <MapXMLRel>

        <!-- Ignora os elementos desnecessarios-->
        <IgnoraElementoRaiz>
            <Elemento Nome="mysqldump"/>
        </IgnoraElementoRaiz>
        <IgnoraElementoRaiz>
            <Elemento Nome="database"/>
        </IgnoraElementoRaiz>
        <IgnoraElementoRaiz>
            <Elemento Nome="table_structure"/>
        </IgnoraElementoRaiz>
        <IgnoraElementoRaiz>
            <Elemento Nome="table_data"/>
            <FilhoRaiz>
                <Elemento nome="row"/>
            </FilhoRaiz>
        </IgnoraElementoRaiz>

        <!-- Mapeamento de Aluno -->
        <Entidade>
            <Elemento Nome="row"/>
            <Tabela Nome="TAluno"/>
            <Propriedade>
                <Elemento Nome="matricula"/>
                <Coluna Nome="matricula"/>
            </Propriedade>
            <Propriedade>
                <Elemento Nome="nome"/>
                <Coluna Nome="nome"/>
            </Propriedade>
            <Propriedade>
                <Elemento Nome="cpf"/>
                <Coluna Nome="cpf"/>
            </Propriedade>
        </Entidade>

    </MapXMLRel>
</XMLparaBD>
```

Conforme pode ser visto, o mapeamento é trivial e após executar a ferramenta com os documentos XML gerados e os arquivos de mapeamento, como já era esperado, as tabelas criadas foram exatamente as mesmas que foram exportadas e podem ser vistas na Figura 11.

5 CONCLUSÕES

5.1 Considerações finais

Este trabalho teve como objetivo o desenvolvimento de uma ferramenta de carga de BD relacionais para fontes de dados XML.

Foi desenvolvido um conjunto de classes e interfaces (API) que permitem que o sistema desenvolvido possa ser chamado por diferentes aplicações. Isto significa que a ferramenta pode ser utilizada diretamente pelo usuário ou, então, em outras aplicações através da API desenvolvida.

Devido à importância e popularidade cada vez maior do XML como padrão na integração de sistemas, muito se tem pesquisado e desenvolvido para facilitar a carga de BDs relacionais a partir de fontes de dados XML. Porém, a grande maioria das soluções disponíveis no mercado é de código fechado e licença comercial. Outras são soluções complexas e difíceis de usar, ou ainda presas a determinados formatos e dependentes de determinados SGBD.

Neste sentido, foi desenvolvido o presente trabalho, que apresenta uma possível solução simples e multi-plataforma para as necessidades de carga de BDs relacionais a partir de dados contidos em documentos XML. O uso da API JDBC para comunicação com o BD faz com que a ferramenta não seja dependente de nenhuma implementação específica de SGBD, podendo ser usada com todos os SGBDs disponíveis no mercado com suporte ao JDBC.

A ferramenta desenvolvida apresenta alguns diferenciais em relação às demais existentes de código aberto. Além de fornecer facilidades para a geração de arquivos de mapeamento, como a geração automática a partir de DTD e XML Schema, ela consegue gerar scripts XSLT que quando executados sobre os documentos XML com os dados, os transforma em scripts SQL com os comandos INSERT necessários. Tais facilidades não foram encontradas em nenhuma das ferramentas de código aberto estudadas no capítulo de trabalhos relacionados.

5.2 Trabalhos futuros

A partir da pesquisa e desenvolvimento realizados neste trabalho, surgem idéias de possíveis trabalhos futuros. Este projeto pode ser estendido de diversas formas. Dentre as principais medidas que podem ser agregadas destacam-se:

a) Desenvolvimento de uma interface gráfica para auxílio no processo de mapeamento. O processo de mapeamento XML para relacional exige que um arquivo de mapeamento seja criado pelo usuário. Apesar da simplicidade da linguagem proposta e das facilidades da ferramenta na geração desses arquivos, a intervenção do usuário muitas vezes torna-se necessária. Para documentos XML grandes, com muitos elementos, este processo pode se tornar cansativo. Neste sentido uma ferramenta visual que facilite ao máximo o processo de mapeamento seria interessante.

b) Desenvolvimento de um mecanismo de engenharia reversa que permita, com as facilidades do JDBC, analisar um esquema relacional existente e gerar automaticamente o mapeamento XML-Relacional a partir de documentos XML de entrada.

c) Desenvolvimento do caminho inverso na transferência de dados, ou seja, a extração de dados de BD relacionais para documentos XML. Essa capacidade tornaria a ferramenta muito útil na troca de informação entre diferentes sistemas, pois a mesma teria capacidade de extrair dados de um banco no formato de documentos XML e posteriormente importar estes dados e inseri-los em diferentes BDs.

d) Simplificar e estender a linguagem de mapeamento. A linguagem de mapeamento, apesar de simples, pode ser simplificada e estendida de forma a permitir novas funcionalidades. Possíveis extensões seriam possibilitar o mapeamento de uma entidade para mais do que uma tabela do BD e tratar elementos mistos. Uma forma de tratar elementos mistos é fazer o mapeamento dos seus elementos textuais simples (ou PCDATA) para uma tabela separada no BD. Como esses PCDATA podem aparecer várias vezes, a ordem pode ser importante. Logo, deve-se criar uma forma de preservar a ordem de ocorrência como, por exemplo, usando uma coluna para guardar a informação de ordem.

REFERÊNCIAS

ALTOVA. **MapForce – Graphical Data Mapping, Conversion, and Integration Tool**. Disponível em: <http://www.altova.com/mapforce.html>. Acesso em 21 de fevereiro de 2011

ALTOVA. **Relational and XML Database Tools**. Disponível em: < http://www.altova.com/xmlspy/xml-database.html >. Acesso em 16 de fevereiro de 2011

BAPTISTA, C; **Banco de Dados - Capítulo 1: Introdução**, 2008. Disponível em: < http://www.dsc.ufcg.edu.br/~baptista/cursos/BDadosI/Capitulo1.pdf>. Acesso em 1 de fevereiro de 2011

BERNING, C; **DBIx::XML::DataLoader**. Disponível em: < http://search.cpan.org/~cberning/DBIx-XML-DataLoader-1.1b/DataLoader.pm >. Acesso em: 11 de fevereiro de 2011

BEZA, O; PATSALA, M; KERAMOPOULUS, E. **Comparison of XML Support in IBM DB2, Microsoft SQL Server, Oracle**, Dpt. Of Information Technology, Thessaloniki, Greece, 2008

BORNHÕVD, A; BUCHMANN , A. **A Generic Load/Extract Utility for Data Transfer between XML Documents and Relational Databases,** TR-DVS99-1, DVS, Germany, 2000

BOURRET, R; **Mapping language DTD,** 2004. Disponível em: <http://www.rpbourret.com/xmldbms/dtds/xmldbms2.dtd> . Acesso em: 10 de março de 2011

BOURRET, R; **Xml –Enabled Databases**, 2010. Disponível em: < http://www.rpbourret.com/xml/ProdsXMLEnabled.htm>. Acesso em 19 de fevereiro de 2011

BOURRET, R; **Xml and Databases,** 2005. Disponível em: < http://www.rpbourret.com/xml/XMLAndDatabases.htm >. Acesso em 4 de fevereiro de 2011.

BOURRET, R; **XML Database products,** 2010. Disponível em: < http://www.rpbourret.com/xml/XMLDatabaseProds.htm#integration >. Acesso em 5 de fevereiro de 2011.

DA SILVA, L, A; **Introdução a SQL,** 2006. Disponível em: < http://www.luizmatos.eti.br/disciplinas/docs/bd/Introducao_SQL.pdf>. Acesso em 3 de fevereiro de 2011

DATADIRECT TECNOLOGIES. **Stylus Studio**. Disponível em: < http://www.stylusstudio.com/>. Acesso em 10 de fevereiro de 2011

EXOLAB.ORG. **The Castor Project**. Disponível em < http://www.castor.org/ >. Acesso em 11 de fevereiro de 2011

ETASOFT. **Extreme Translator**. Disponível em: < http://www.xtranslator.com/et.htm>. Acesso em: 17 de fevereiro de 2011

FURTADO, M; **XML - Extensible Markup Language.** Disponível em: <http://www.gta.ufrj.br/grad/00_1/miguel/>. Acesso em 3 de fevereiro de 2011

HAROLD, R; MEANS, W. **XML in a Nutshell,** 3a edição, O'Reilly Media, 2004

HIT SOFWARE. **Allora, Graphical DatabaseXML Mapping and Transformation**. Disponível em: < http://www.hitsw.com/products_services/xmlplatform.html>. Acesso em 22 de fevereiro de 2011

IBM**. DB/XML Transform™.** Disponível em: < http://www.treehouse.com/DBXMLTransform.shtml>. Acesso em 17 de fevereiro de 2011

MACHADO, L; **O que representa os DBMS relacionais?,** 2007

MATTOSO, M; **Introdução a Bancos de Dados – o modelo relacional,** 2008**.** Disponível em: < http://www.cos.ufrj.br/~marta/BdRel.pdf>. Acesso em 18 de Fevereiro de 2011.

MICROSOFT. **Persisting Records in XML Format**. Disponível em: < http://msdn.microsoft.com/en-us/library/ms681538.aspx>. Acesso em: 10 de fevereiro de 2011

MICROSOFT. **SQLXML**. Disponível em: < http://msdn.microsoft.com/en-us/library/aa286527.aspx>. Acesso em: 5 de fevereiro de 2011

NAVATHE; ELMASRI; **Fundamentals of DATABASE SYSTEMS,** 4a edição. Addison Wesley , 2003.

ODONATA**. XQuare Bridge: extending relational databases with XQuery and XML-relational mapping**
Disponível em: < http://xquare.ow2.org/bridge/index.html>. Acesso em 11 de fevereiro de 2011

OKERKE, G; IKEJA, L; **Database Management System.** Disponível em: <http://www.nou.edu.ng/noun/NOUN_OCL/pdf/pdf2/MBA%20758%20Database%20Management%20System.pdf>. Acesso em: 2 de fevereiro de 2011

ORACLE**. Oracle XML Developer's Kit (XDK) Java - 10.2.0.2.0 LINUX Production**. Disponível em: < http://www.oracle.com/technetwork/database/features/xmldb/java-utilsoft-087831.html>. Acesso em 12 de fevereiro de 2011.

QUEIROZ, J; **SGBD: O que é?** Disponível em: <http://espacoinfo.net/o-que-e-sgbd-bd-ii>. Acesso em: 1 de fevereiro de 2011

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. 2a. edição. WCB McGraw-Hill, 1999.`

SILVA, Cassandra Ribeiro de O. e. **Metodologia e Organização do Projeto de Pesquisa** (Guia Prático). Fortaleza, 2004. Disponível em: < http://www.professormanueljunior.com/geral/arquivos/MANUAL%20DE%20METODOLOGIA.pdf>. Acesso em: 7 de abril de 2011.

SILVA, J; **SQL (MySQL)**, 2007**.** Disponível em: < http://www.ifpiparnaiba.edu.br/index.php?option=com_docman&task=doc_download&gid=24&Itemid=79>. Acesso em 3 de fevereiro de 2011

SKYHAWK SYSTEMS. **Connect XML-2-DB**. Disponível em: < http://www.skyhawksystems.com/ >. Acesso em: 22 de fevereiro de 2011.

SOFTRUS. **Db To Xml**. Disponível em: < http://www.soft-r-us.com/dbtoxml.asp > Acesso em: 16 de fevereiro de 2011

STEWART, G; **Osage - Persistence Plus XML**. Disponível em: < http://osage.sourceforge.net/>. Acesso em12 de fevereiro de 2011.

VIGNATTI, A; **Introdução a DTD,** 2009. Disponível em: < http://www.dainf.ct.utfpr.edu.br/~vignatti/courses/if54e/Introducao_a_DTD.pdf >. Acesso em 03 de fevereiro de 2011.

W3C; **Extensible Markup Language (XML)**, 2011. Disponível em: < http://www.w3.org/XML/>. Acesso em 12 de Fevereiro de 2011

W3CSCHOOLS; **DTD Tutorial**, Disponível em < http://www.w3schools.com/dtd/default.asp >. Acesso em 13 de fevereiro de 2011

W3CSCHOOLS; **Introduction to SQL**, Disponível em < http://www.w3schools.com/sql/sql_intro.asp >. Acesso em 13 de fevereiro de 2011

W3CSCHOOLS; **XML Tutorial,** Disponível em: < http://www.w3schools.com/xml/default.asp>. Acesso em: 12 de fevereiro de 2011

W3CSCHOOLS; **XML Schema Tutorial,** Disponível em: < http://www.w3schools.com/schema/default.asp>. Acesso em: 12 de fevereiro de 2011

W3CSCHOOLS; **XPATH,** Disponível em: < http://www.w3schools.com/xpath/default.asp>. Acesso em: 14 de fevereiro de 2011

W3CSCHOOLS; **XSLT,** Disponível em: < http://www.w3schools.com/xsl/ >. Acesso em: 14 de fevereiro de 2011

APÊNDICE A – PRINCIPAIS COMANDOS OFERECIDOS PELA FERRAMENTA

- Comando que faz a carga do BD relacional. Recebe o caminho para o arquivo de mapeamento (arquivo .map), o(s) documento(s) XML e o arquivo de configuração com informações de acesso ao BD através do JDBC:

java XmlBDTool –transf <arquivo_map> <lista_arquivos_xml> <arquivo_conf>

- Comando que permite gerar um script XSL a parir do arquivo de mapeamento:

java XmlBDTool –xslt <arquivo_map> <arquivo_xsl_saida>

- Comando que permite gerar um arquivo de mapeamento (arquivo .map) a partir de um arquivo DTD ou XSD:

java XmlBDTool –geramap <arquivo_dtd_ou_xsd> <arquivo_map_saida>

- Comando que gera um script SQL com comandos CREATE TABLE a partir do mapeamento:

java XmlBDTool –sql <arquivo_map> <arquivo_sql_saida>

- Comando que gera um script SQL com comandos INSERT necessários para fazer a carga do BD. Não envia os comandos através do JDBC, apenas gera o script:

java XmlBDTool –sql <arquivo_map> <lista_arquivos_xml> <arquivo_sql_saida>

- Comando que envia comandos INSERT e/ou CREATE TABLE para o BD através do JDBC a partir de um script SQL:

java XmlBDTool –transf <arquivo_sql> <arquivo_conf>

- Comando que gera um script SQL a partir de um script XSL e do documento XML com os dados:

java XmlBDTool –sql <arquivo_xsl> <doc_xml> <arquivo_sql_saida>

APÊNDICE B – DTD DA LINGUAGEM DE MAPEAMENTO

```
<!ELEMENT XMLparaBD (Opcoes*, MapXMLRel)>
<!ATTLIST XMLparaBD
        Versao CDATA #FIXED "1.0">

<!ELEMENT Opcoes (TrataStringVaziaComoNulo?, Namespace*)>


<!ELEMENT TrataStringVaziaComoNulo EMPTY>

<!ELEMENT Namespace EMPTY>
<!ATTLIST Namespace
        Prefixo NMTOKEN #REQUIRED
        URI CDATA #REQUIRED>

<!ELEMENT MapXMLRel (IgnoraElementoRaiz*, Entidade+)>


<!ELEMENT IgnoraElementoRaiz (Elemento, FilhoRaiz+)>

<!ELEMENT FilhoRaiz (Elemento)>

<!ELEMENT Entidade (Elemento, Tabela, Propriedade*, EntidadeRelacionada*)>

<!ELEMENT Propriedade ((Atributo |  Elemento), Coluna)>


<!ELEMENT EntidadeRelacionada (Elemento, (ChavesJuncao | TabelaAssociativa))>
<!ATTLIST EntidadeRelacionada
        tipoRelacionamento (Direto | N-N) #REQUIRED>

<!ELEMENT ChavesJuncao (ChaveUnica, ChaveEstrangeira)>
<!ATTLIST ChavesJuncao
        ChaveEntidadePai (Unica | Estrangeira) #REQUIRED>

<!ELEMENT ChaveUnica (Coluna+)>
<!ATTLIST ChaveUnica
        GerarAutomaticamente (S | N) #REQUIRED>

<!ELEMENT ChaveEstrangeira (Coluna+)>

<!ELEMENT TabelaAssociativa (Coluna+)>
<!ATTLIST TabelaAssociativa
        %XMLName; #REQUIRED>
        elemento NMTOKEN #IMPLIED>
        idref NMTOKEN #IMPLIED>
        idrefs NMTOKEN #IMPLIED>
```

```
<!ENTITY % XMLName "Nome NMTOKEN #REQUIRED">

<!ELEMENT Elemento EMPTY>
<!ATTLIST Elemento
        %XMLName;>
         vazio CDATA "N">

<!ELEMENT Atributo EMPTY>
<!ATTLIST Atributo
        %XMLName;>

<!ENTITY % DatabaseName "Nome CDATA #REQUIRED">

<!ELEMENT Tabela EMPTY>
<!ATTLIST Tabela
        %DatabaseName;>

<!ELEMENT Coluna EMPTY>
<!ATTLIST Coluna
        %DatabaseName;
         valor NMTOKEN #IMPLIED>
```

APÊNDICE C – ARTIGO

Uma ferramenta para carga de bancos de dados relacionais a partir de fontes de dados XML

Edwaldo Monteiro[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis, SC – Brasil

edymont@inf.ufsc.br

Abstract. *With the evolution of the Internet, information systems and e-commerce, the exchange of information between different applications and systems has become an increasingly frequent requirement. Thanks to its great flexibility and portability, XML has been widely accepted in the past years as the standard for representing, transferring and manipulating data in many applications. Nowadays, XML plays a fundamental role in integrating heterogeneous systems such as in the exchange of information between two different database systems. Relational database systems are the most common these days, and a lot of the business information is stored in organizations according to the relational model of data. This work consists of developing a tool for performing the load of relational databases from data contained in XML documents. The tool was implemented using the JAVA language and its core technologies, which make it multi-platform and DBMS independent.*

Resumo. *Com a evolução da internet, dos sistemas de informação e do comércio eletrônico, a troca de informações entre diferentes aplicações e sistemas se tornou um requisito cada vez mais frequente. Graças à sua grande flexibilidade e portabilidade, o XML nos últimos anos vem sendo cada vez mais aceito como um padrão para representar, transferir e manipular dados em diversas aplicações. Hoje em dia, o XML desempenha um papel fundamental na integração de sistemas heterogêneos como, por exemplo, na troca de informações entre sistemas de banco de dados diferentes. Os sistemas de banco de dados relacionais são os mais utilizados na atualidade, sendo que grande parte da informação de negócio das empresas e organizações, está armazenada de acordo com o modelo relacional de dados. Este trabalho consiste no desenvolvimento de uma ferramenta que possibilita a carga de bancos de dados relacionais diversos a partir de dados contidos em documentos XML. A ferramenta foi implementada usando a linguagem JAVA e suas principais tecnologias, tornando-a multi-plataforma e independente de SGBD..*

1. Introdução

O XML (*Extensible Markup Language*) desempenha um papel cada vez mais importante como sendo um formato genérico de troca de dados, sobressaindo-se sobre

outros formatos (W3C; 2011). O seu suporte ao UNICODE torna-o altamente portável e capaz de representar praticamente qualquer *string*. Além disto, o uso de *tags* para rotular dados faz com que documentos XML sejam auto-descritivos, podendo ser facilmente lidos por humanos e ao mesmo tempo usados por diferentes aplicações. O formato padronizado de um documento XML possibilita que o mesmo possa ser lido por qualquer aplicação.

Com a evolução da internet, dos sistemas de informação e do comércio eletrônico, a troca de informações entre diferentes aplicações se tornou um requisito cada vez mais frequente, sendo necessário prover interoperabilidade entre sistemas heterogêneos.

Ainda de acordo com a W3C (2011), o principal uso do XML é de transferência de dados entre aplicações distintas e em ambientes heterogêneos. Antes da "era XML", as empresas que necessitavam de transferência de dados, utilizavam o EDI (*Electronic Data Interchange*), onde um grande número de transações era comunicado em arquivos em lote. Obviamente as empresas tinham de negociar previamente padrões pouco flexíveis para os arquivos em lote. Hoje em dia, com a flexibilidade e portabilidade do XML, uma aplicação pode gerar documentos XML a partir de dados armazenados em um banco de dados (BD), e estes documentos podem facilmente ser lidos posteriormente por outra aplicação, e os dados contidos nos mesmos podem ser extraídos e armazenados em um modelo de dados possivelmente diferente do BD inicial.

Os BDs relacionais são os bancos de dados mais utilizados na atualidade, sendo que grande parte da informação de negócio das empresas e organizações está armazenada de acordo com o modelo relacional de dados. A troca de dados entre documentos XML e bancos relacionais é algo possível e muito importante na integração de sistemas de informação. Neste sentido, existem softwares desenvolvidos especificamente para realizar a carga de BD relacionais a partir de dados contidos em documentos XML, e também para realizar a extração de dados armazenados no BD e armazená-los em documentos XML.

Alguns dos principais Sistemas de Gerenciamento de Banco de Dados (SGBD) relacionais existentes no mercado atualmente disponibilizam extensões para extrair dados de documentos XML e armazená-los no BD e vice versa. Porém, na sua grande maioria, são soluções comerciais pagas, de código fechado, limitadas e pouco flexíveis. Geralmente exigem um formato específico ou proprietário para os documentos XML e obviamente a transferência só pode ser feita através do SGBD em questão (BEZA et al; 2008). Algumas das ferramentas mais poderosas disponíveis, que permitem armazenar dados contidos num documento XML em um BD relacional também são soluções proprietárias, de código fechado e licença comercial e a maioria se restringe a um conjunto limitado de SGBD. Das soluções gratuitas disponíveis, a maioria é complexa e possui várias funcionalidades além da carga de BDs relacionais a partir de documentos XML, o que se torna um inconveniente para o usuário quando o mesmo quer uma solução simples e de uso fácil, que o permita armazenar dados em BDs relacionais a partir de documentos XML.

O objetivo geral deste trabalho consiste no desenvolvimento de uma ferramenta de uso fácil, independente de plataforma e de código aberto, para carga de BDs

relacionais diversos a partir de dados contidos em documentos XML. O restante deste artigo está organizado da seguinte forma: Seção 2 apresenta alguns trabalhos relacionados; Seção 3 descreve as funcionalidades da ferramenta; Seção 4 apresenta o mapeamento XML-Relacional; Seção 5 apresenta o projeto da ferramenta desenvolvida. Seção 6 mostra um exemplo completo e por fim, a Seção 7 conclui o artigo.

2. Trabalhos relacionados

Com a crescente popularização do XML como um meio eficiente de troca de dados entre aplicações e integração de sistemas, e com o surgimento de novas tecnologias e ferramentas poderosas complementares ao XML, muito se tem desenvolvido no sentido de facilitar a troca de dados entre documentos XML e diferentes BDs.

Segundo Bourret (2010), os softwares que permitem fazer a transferência de dados entre documentos XML e BDs relacionais podem ser divididos em quatro categorias diferentes. São elas:

a) Middlewares;

b) IDE (*Integrated Development Environment*) e editores;

c) BDs com suporte a XML;

d) Softwares de integração de dados.

A ferramenta desenvolvida se enquadra na categoria de Middlewares. Um middleware é um software de terceiros, que é usado por aplicações para fazer a carga de BDs a partir de documentos XML. A aplicação invoca o middleware, que por sua vez faz todo o trabalho de carga do BD. A maioria dos middlewares acessa os dados em BD relacionais através de tecnologias como o ODBC (Open Data Base Connectivity) e JDBC (Java Database Connectivity).

O Quadro 1 faz um resumo dos principais middlewares disponíveis no mercado e mostra algumas diferenças básicas entre os mesmos como a licença de uso, o tipo de BD que suportam e o sentido da transferência de dados que é suportado, podendo ser do BD para um documento XML ou vice-versa.

Produto	**Desenvolvedor**	**Licença de uso**	**Tipo BD**	**BD para XML**	**XML para BD**
ADO	Microsoft	Comercial	Relacional	Sim	Sim
Allora	HiT Software	Comercial	Relacional	Sim	Sim

Altova MapForce	Altova	Comercial	Relacional	Sim	Sim
Castor	exolab.org	Open Source	Relacional	Sim	Sim
Connect XML-2-DB	Skyhawk Systems	Comercial	Relacional	Não	Sim
DbToXml	SoftRUs	Comercial	Relacional	Sim	Sim
DBIx::XML::DataLoader	Christopher Berning	Open Source	Relacional	Não	Sim
DB/XML Transform	IBM	Comercial	Relacional	Sim	Sim
Extreme Translator	Etasoft	Comercial	Relacional	Sim	Sim
Oracle XML Developer's Kit (XDK)	Oracle	Freeware	Relacional	Sim	Sim
Osage	George Stewart, et al	Open Source	Relacional	Sim	Sim
dtd2sql e xml2sql	David Mertz	Open Source	Relacional	Sim	Sim
xmlToSql	Jim Kent	Freeware	Relacional	Sim	Sim
SQLXML	Microsoft	Comercial	Relacional	Sim	Sim
XQuare Bridge	Odonata	Open Source	Relacional, XML	Sim	Sim

Quadro 1 – Comparação entre alguns dos principais middlewares de transferência de dados entre documentos XML e BDs relacionais

3. Funcionalidades

A ferramenta desenvolvida agrega as seguintes funcionalidades:

a) Possibilita a carga de qualquer BD relacional cujo SGBD suporte a API JDBC, a partir de dados contidos em um documento XML;
b) Gera um arquivo XSL contendo comandos XSLT capazes de transformar o documento XML com os dados, em um script SQL com os comandos INSERT;
c) Disponibiliza uma linguagem simples para mapeamento XML-Relacional, com a qual é possível criar arquivos de mapeamento (arquivo .map);
d) Gera automaticamente o mapeamento XML-Relacional a partir de esquemas XML (DTD e XML Schema);
e) Gera automaticamente comandos CREATE TABLE para as tabelas envolvidas na transferência de dados;
f) Dá suporte a namespaces, geração automática de chaves únicas e transações para SGBD que as suportam.

O funcionamento básico da ferramenta consiste em receber como entradas o documento XML com os dados e o arquivo de mapeamento (.map) com as regras do mapeamento, escritas usando a linguagem de mapeamento. Tal arquivo é então passado ao módulo gerador de mapeamento onde é processado por uma das classes geradoras e um objeto Mapeamento é gerado. Esse objeto pode ser serializado para uso posterior ou passado, juntamente com o(s) documentos(s) XML, ao módulo transferidor de dados, que se encarrega de fazer a conversão dos dados de acordo com o objeto Mapeamento recebido e a posterior inserção dos mesmos no BD relacional. O transferidor de dados ainda tem a capacidade de gerar automaticamente chaves únicas caso seja necessário no momento da inserção no BD.

Além disso, é possível gerar um script SQL com os comandos CREATE TABLE que pode ser usado para gerar as tabelas envolvidas na transferência de dados. Note-se que um mesmo objeto Mapeamento pode ser utilizado mais do que uma vez para a carga do BD e para gerar o esquema relacional.

Outra entrada possível para o sistema é um arquivo DTD ou XML Schema contendo o esquema do documento XML. Neste caso, um objeto Mapeamento é gerado automaticamente pelas classes geradoras e então se pode gerar o esquema relacional, salvar o mapeamento no sistema de arquivos (em forma de um arquivo .map) ou fazer a carga do BD a partir de dados contidos em um determinado documento XML.

O arquivo XSD ou DTD pode ser o arquivo de validação para o documento XML que contém os dados ou ainda pode ser obtido com auxílio de ferramentas como o db2xsd, que são capazes de gerar um XML Schema ou DTD a partir de um esquema relacional ou de um documento XML.

O sistema pode ainda gerar um arquivo XSL a partir do mapeamento. O arquivo contém comandos XSLT, e quando executado por um transformador XSLT, sobre o documento XML, gera como saída um script SQL com os comandos INSERT. O arquivo XSL gerado pode ser usado em outras aplicações ou processado pelo próprio módulo XSLT da ferramenta.

Por fim, os scripts SQL gerados tanto pelo módulo XSLT como pelo módulo transferidor de dados podem ser processados diretamente pelo módulo JDBC da ferramenta e os comandos INSERT ou CREATE TABLE enviados para o SGBD.

4. O mapeamento XML-Relacional

O modelo de dados de um documento XML é hierárquico. Isto significa que os dados estão organizados numa estrutura hierárquica de árvore, o que difere totalmente do modelo relacional, que se baseia em tabelas e relacionamento entre as mesmas através do uso de chaves. Portanto, para que seja possível fazer a carga de um BD relacional a partir de um documento XML, torna-se necessário fazer a conversão da estrutura hierárquica para a estrutura de tabelas.

Com esta finalidade, é oferecida uma linguagem baseada em XML que possibilita descrever o mapeamento dos dados XML para as tabelas do BD relacional.

Já os esquemas XML (DTD e XML Schema) podem ser mapeados automaticamente pela ferramenta, para o esquema relacional, gerando arquivos de mapeamento que podem então ser editados pelo usuário.

Na ferramenta proposta, o mapeamento XML-relacional é feito através de dois conceitos importantes: **entidade** e **propriedade.** Nesta abordagem, os elementos de um documento XML podem ser vistos tanto como entidades como propriedades. Os elementos complexos, ou seja, elementos que contêm outros elementos são geralmente vistos como entidades e posteriormente serão mapeados para tabelas do BD relacional. Já os elementos simples de texto ou do tipo PCDATA (*Parsed Character Data*) são vistos como propriedades de alguma entidade. Quando um elemento de um documento XML é visto como uma propriedade, tal propriedade pertence à entidade correspondente do seu elemento pai (ascendente direto) e será mapeado para uma coluna da tabela correspondente.

Além disto, quando um elemento é considerado como sendo uma entidade, os seus atributos e elementos texto simples são vistos como propriedades da entidade em questão. Através da linguagem de mapeamento oferecida pela ferramenta, é possível especificar como será visto cada elemento do documento XML e o relacionamento entre os mesmos.

Uma vez definidas as entidades envolvidas, o mapeamento se torna direto, sendo que cada entidade corresponde a uma tabela, as suas propriedades correspondem às colunas da tabela e os relacionamentos entre-entidades correspondem a pares de chaves (única e estrangeira) ou tabelas associativas.

4.1 A linguagem de mapeamento

A linguagem de mapeamento oferecida pela ferramenta não é mais do que um esquema XML com descrição de *tags* bem específicas e com significados bem definidos. Um arquivo de mapeamento (arquivo MAP) é um documento XML bem formado e que respeita as regras da linguagem descritas pelo esquema XML. O esquema XML com a linguagem de mapeamento é então usado para validar os arquivos MAP, antes de o sistema gerar o objeto Mapeamento correspondente.

A seguir é mostrado o formato do arquivo de mapeamento de forma simplificada e em seguida cada tag da linguagem XML é explicada em maiores detalhes.

```
<MapXMLRel>
        <Opcoes>
        ...
</Opcoes>
<Mapeamentos>
        ...
        < Entidade >
        < /Entidade>
        ...
</Mapeamentos >
</MapXMLRel>
```

O elemento ‘MapXMLRel’ é o elemento raiz do arquivo de mapeamento, ou seja, todos os outros elementos são descendentes do mesmo.

O elemento ‘Opcoes’ é opcional e serve para especificar algumas opções como o uso de namespaces e como as strings vazias no documento XML devem ser tratadas. Contém o elemento ‘NameSpace’ e o ‘TrataStringVaziaComoNulo’. A estrutura do elemento é a seguinte:

```
<Opcoes>
        < TrataStringVaziaComoNulo/>
        <Namespace Prefixo=”algumPrefixo” URI=”algumaURI”/>
        <Namespace Prefixo=”algumPrefixo2” URI=”algumaURI2”/>
        ...
        <Namespace Prefixo=”algumPrefixoN” URI=”algumaURIN”/>
</Opcoes>
```

Tanto o elemento ‘TrataStringVaziaComoNulo’ como o ‘Namespace’ são opcionais. Se o elemento ‘TrataStringVaziaComoNulo’ estiver presente, as strings vazias encontradas no documento XML que contém os dados, são tratadas como nulas (NULL), caso contrário são tratadas como strings quaisquer.

O elemento ‘NameSpace’ pode ou não aparecer e pode se repetir N vezes. Os seus dois atributos ‘Prefixo’ e ‘URI’ indicam o prefixo de um namespace que será usado

no documento XML com os dados e a URI do namespace, respectivamente. Este elemento é usado na geração de scripts, como o XSLT.

O elemento ‘Mapeamentos’ é o elemento que contém um ou mais elementos ‘Entidade’, que por sua vez contém a informação de como mapear um determinado elemento do documento XML visto como entidade para o BD relacional. Além disso, o elemento ‘Mapeamentos’ pode conter um elemento opcional chamado ‘IgnoraElementoRaiz’. A sua estrutura é a seguinte:

```
<Mapeamentos>
        <IgnoraElementoRaiz>
        ...
        </IgnoraElementoRaiz>
        <Entidade>
        ...
        </Entidade>
        ...
</Mapeamentos >
```

O Elemento ‘IgnoraElementoRaiz’ permite especificar que o elemento raiz do documento XML deve ser ignorado no mapeamento e na transferência de dados. A estrutura do XML exige que exista um elemento raiz em cada documento. Porém, tal elemento pode existir apenas para suprir essa exigência e não ter qualquer relação com o BD relacional. O elemento ‘IgnoraElementoRaiz’ é opcional e pode aparecer várias vezes visto que podemos ter um documento com várias instâncias sem nenhuma relação. O elemento filho ‘FilhoRaiz’ permite especificar quais os elementos filhos da raiz serão mapeados para o BD e pode aparecer várias vezes. A estrutura é:

```
<IgnoraElementoRaiz>
        <Elemento nome=”Pessoas”/>
        < FilhoRaiz >
            <Elemento nome=”Pessoa”/>
        < /FilhoRaiz>
</IgnoraElementoRaiz>
```

O elemento ‘Entidade’ é o elemento que contém as informações do mapeamento de cada elemento do documento XML visto como entidade, para o BD relacional. Ele permite indicar para qual tabela o elemento será mapeado e especificar o mapeamento das suas propriedades e relacionamentos com outros elementos. A estrutura é:

```
<Entidade>
        <Elemento nome=”nomeElemento”/>
        <Tabela nome=”NOMETABELA”/>
        <Propriedade>
            ...
```

```
        </Propriedade>
        <EntidadeRelacionada>
                ...
        </EntidadeRelacionada>
        <IgnoraElemento>
                ...
        </IgnoraElemento>
        ...
</Entidade>
```

O elemento de nome ‘Elemento’ especifica qual o elemento a ser mapeado como entidade. O atributo ‘nome’ do elemento ‘Tabela’ especifica o nome da tabela no BD para a qual a entidade será mapeada. O elemento ‘Propriedade’ que é opcional e pode se repetir várias vezes especifica quais atributos e elementos simples serão mapeados para quais colunas da tabela correspondente à entidade sendo mapeada. O atributo opcional ‘vazio’ do elemento ‘Elemento’ indica se trata-se de um elemento vazio e por omissão o seu valor é “N”. Caso o valor seja “S”, a propriedade é mapeada para uma coluna de tipo *BOOLEAN*. Se o elemento estiver presente no documento XML, o valor da coluna será *true*, caso contrário, *false.* A estrutura é a seguinte:

```
<Propriedade>
        <Atributo nome=”nomeAtributo” />
        <!— ou  <Elemento nome=”nomeElemento” vazio=”S ou N”/> -->
        <Coluna nome =”nomeColuna”/>
</Propriedade>
```

O elemento ‘EntidadeRelacionada’ que também é opcional e pode aparecer várias vezes especifica os relacionamentos da entidade mapeada com outras entidades, bem como as chaves única e estrangeira utilizadas no BD para a junção das tabelas correspondentes ou ainda a tabela associativa utilizada no BD para relacionar as tabelas. O elemento ‘EntidadeRelacionada’ possui um atributo obrigatório de nome ‘tipoRelacionamento’ que permite especificar o tipo de relacionamento entre as duas entidades (‘Direto’ ou ‘N-N’). O tipo ‘Direto’ representa os relacionamentos ‘1-1’ e ‘1-N’ pois os mesmos são tratados de forma igual. Ambos são relacionamentos em que uma das tabelas envolvidas armazena uma chave estrangeira que referencia uma entidade da outra tabela. Nestes casos a estrutura é a seguinte:

```
<EntidadeRelacionada tipoRelacionamento=”Direto”>
        <Elemento nome=”nomeElemento”/>
        <ChavesJuncao ChaveEntidadePai=”Unica”>
                <ChaveUnica gerarAutomaticamente=”N”>
                        <Coluna nome =”nomeColuna”/>
                </ChaveUnica>
                <ChaveEstrangeira>
```

```
                <Coluna nome =”nomeColuna”/>
            </ChaveEstrangeira>
        </ChavesJuncao>
</EntidadeRelacionada>
```

O elemento ‘Elemento’ especifica qual o elemento que se relaciona com o elemento sendo mapeado. O atributo ‘ChaveEntidadePai’ indica se a chave presente na tabela correspondente à entidade pai (lembrando que as declarações de entidades relacionadas estão aninhadas dentro da declaração de uma entidade) é única ou estrangeira. Isso possibilita mais flexibilidade na formação dos documentos XML. A mesma informação pode ser estruturada de formas diferentes.

Os elementos ‘ChaveEstrangeira’ e ‘ChaveUnica’ permitem especificar as chaves usadas no BD para fazer a junção das tabelas. O elemento ‘Coluna’ pode se repetir mais de uma vez caso as chaves sejam compostas por mais de uma coluna. O atributo ‘gerarAutomaticamente’ indica se a chave única deve ser gerada pela ferramenta.

Já em caso de relacionamento “N-N”, em vez do elemento ‘ChavesJuncao’, está presente o elemento ‘TabelaAssociativa’, que especifica detalhes da tabela associativa usada no banco para o relacionamento N-N. A estrutura é a seguinte:

```
<EntidadeRelacionada tipoRelacionamento=”N-N”>
        <Elemento nome=”nomeElemento”/>
        <TabelaAssociativa nome=”NOMETABELAASSOC” elemento=”nelemento”
idref=”atributoIDREF” idrefs=”atributoIDREFS”/>
                <Coluna nome=”COLUNA1” valor=”TabPai.Coluna”/>
                <Coluna nome=”COLUNA2” valor=”TabFilho.Coluna”/>
                <Coluna .../>
                ...
        </ TabelaAssociativa>
</EntidadeRelacionada>
```

O elemento ‘Elemento’ especifica o elemento com o qual a entidade sendo mapeada se relaciona através de uma tabela associativa.

O atributo ‘nome’ do elemento ‘TabelaAssociativa’ especifica o nome da tabela associativa no BD. O atributo ‘elemento’ é opcional e pode especificar qual o elemento simbólico no documento XML que contem as informações que serão mapeadas para a tabela associativa através de uma expressão XPath. Os atributos idref e idrefs são opcionais e especificam que atributo de valor único no documento XML aponta para o elemento sendo mapeado na seção ‘EntidadeRelacionada’.

Por fim, o elemento ‘<IgnoraElemento>’ serve para informar que um determinado elemento no documento XML existe para fins de organização dos dados mas não deve ser mapeado para uma tabela do BD. Neste caso, o elemento é ignorado e suas propriedades são mapeadas como sendo propriedades da entidade pai.

5. Projeto da ferramenta

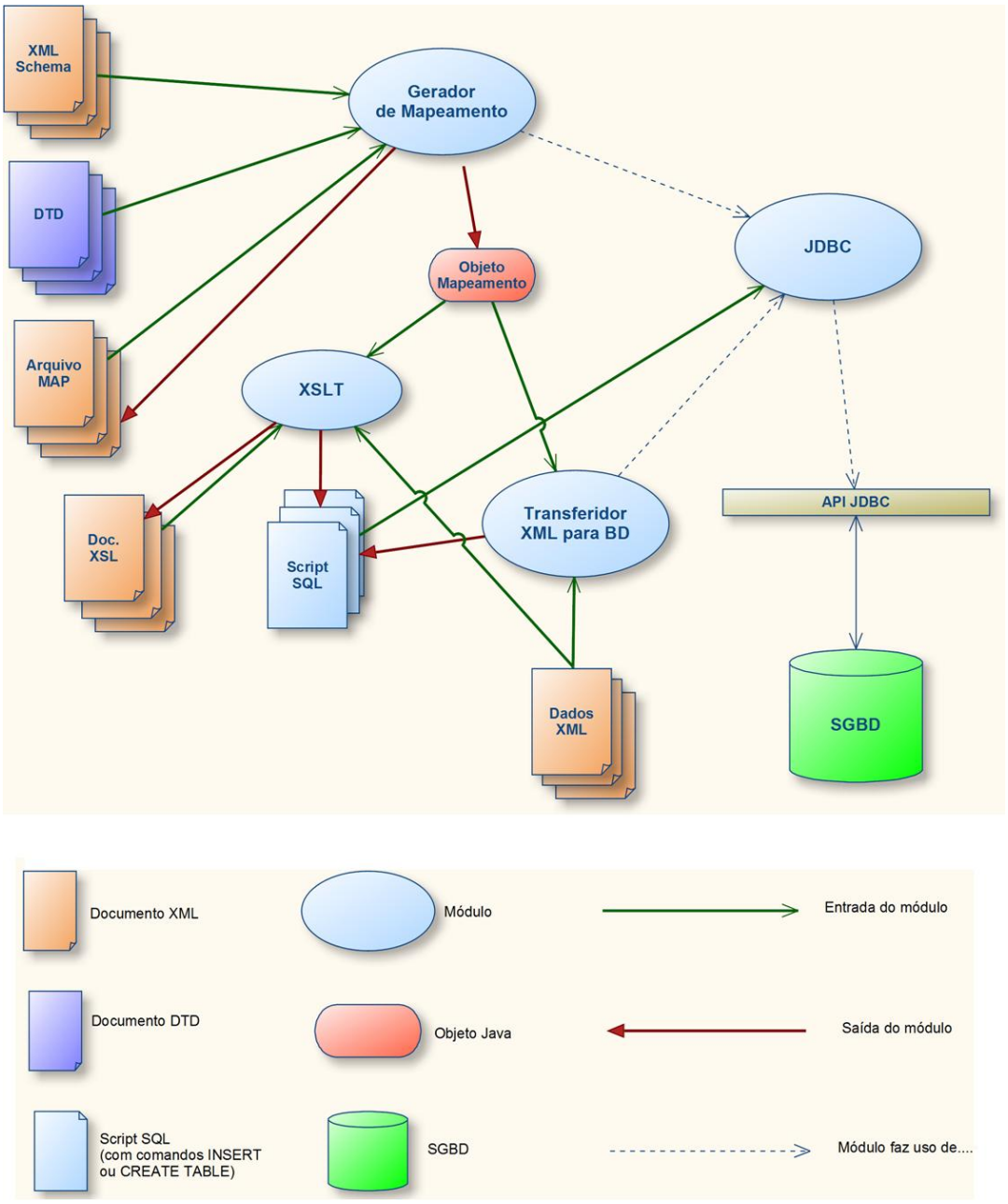

Figura 1: Arquitetura da ferramenta desenvolvida na visão de módulos

Conforme ilustrado na Figura 1, são quatro os módulos desenvolvidos:

a) Módulo gerador de mapeamento;
b) Módulo XSLT;
c) Módulo transferidor de dados XML para o BD;
d) Modulo JDBC.

O módulo **gerador de mapeamento** é o módulo responsável por processar os arquivos de mapeamento (arquivos .map) e criar objetos da classe Mapeamento a partir de tais arquivos. O módulo ainda tem a capacidade de processar esquemas XML (DTD e XML Schema) e a partir destes, gerar arquivos de mapeamento. Os arquivos gerados podem, então, ser editados pelo usuário e submetidos novamente ao módulo gerador, para a criação de objetos da classe Mapeamento. Caso seja especificado pelo usuário, o módulo gerador faz uso do módulo JDBC para validar o mapeamento gerado, verificando a existência das tabelas e colunas especificadas no mapeamento, no BD, bem como o tipo de dado das colunas, etc.

O objeto da classe Mapeamento gerado pelo módulo gerador de mapeamento, serve de entrada tanto para o módulo XSLT como para o módulo transferidor de dados.

O **módulo XSLT** é o módulo responsável por gerar, a partir do objeto da classe Mapeamento recebido como entrada, arquivos XSL com comandos XSLT capazes de transformar o documento XML, que contém os dados, em um script SQL com comandos INSERT necessários para fazer a carga do BD. Este módulo tem ainda a capacidade de executar a transformação e gerar scripts SQL a partir de arquivos XSL já gerados e dos documentos XML com os dados.

O **módulo transferidor de dados XML para BD** é o módulo responsável por fazer a carga do BD a partir dos dados contidos em documentos XML. Recebendo como entrada um objeto da classe Mapeamento, gerado pelo módulo gerador de mapeamentos e um documento XML com os dados, este último é então processado e, a partir dos dados contidos no mesmo, é feita a carga do BD. Este módulo tem ainda a capacidade de gerar scripts SQL com os comandos INSERT correspondentes ou scripts SQL com comandos CREATE TABLE para criar as tabelas no BD necessárias para a carga do mesmo. O módulo transferidor faz uso do módulo JDBC para enviar os comandos SQL para o SGBD e para obter informações do BD.

O módulo JDBC é o módulo responsável pela comunicação com o BD através da API JDBC. Outros módulos fazem uso deste módulo para enviar comandos SQL para o SGBD ou para obter alguma informação do BD. O módulo JDBC ainda tem a capacidade de receber arquivos SQL com comandos INSERT ou CREATE TABLE e enviá-los diretamente para o SGBD.

O código da ferramenta encontra-se dividido em quatro pacotes Java:

a) **Mapeamento**: contendo todas as classes envolvidas no mapeamento XML-Relacional;

b) **Geradores**: contendo as classes utilizadas para gerar o objeto Mapeamento a partir de um arquivo de mapeamento, um DTD ou um XML Schema;

c) **Transferidor**: contendo as classes necessárias para fazer a carga do BD, bem como para gerar o esquema relacional e gerar o arquivo XSL;

d) **Utils**: contendo utilitários e classes de uso geral

A Figura 2 ilustra os 4 pacotes citados e as principais classes pertencentes a cada um.

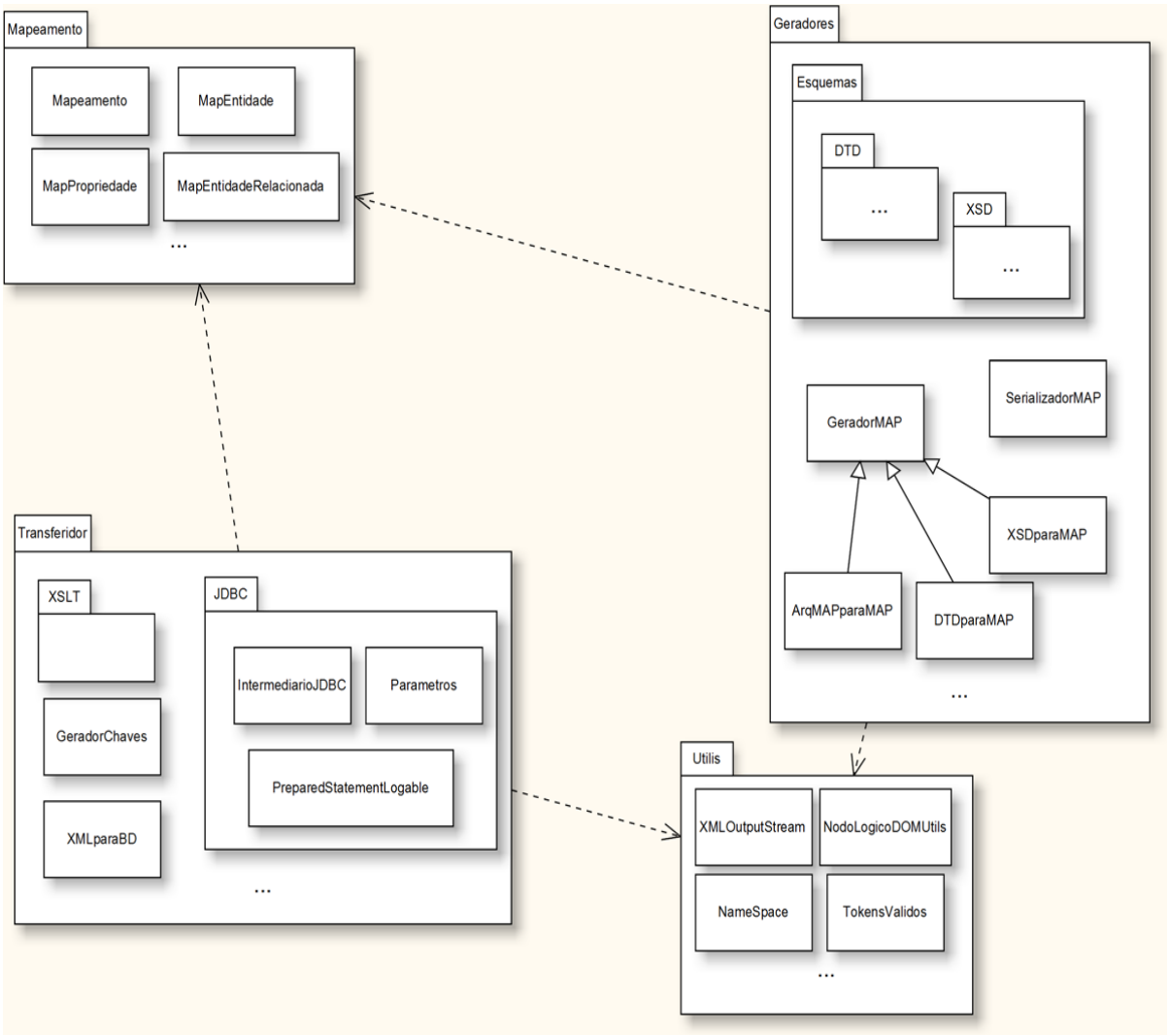

Figura 2: diagrama de pacotes

Na Figura 3, é apresentado um diagrama com as principais classes do sistema. Devido à grande quantidade de classes, apenas as principais e de uso geral são apresentadas. As demais são classes auxiliares e de uso restrito da implementação.

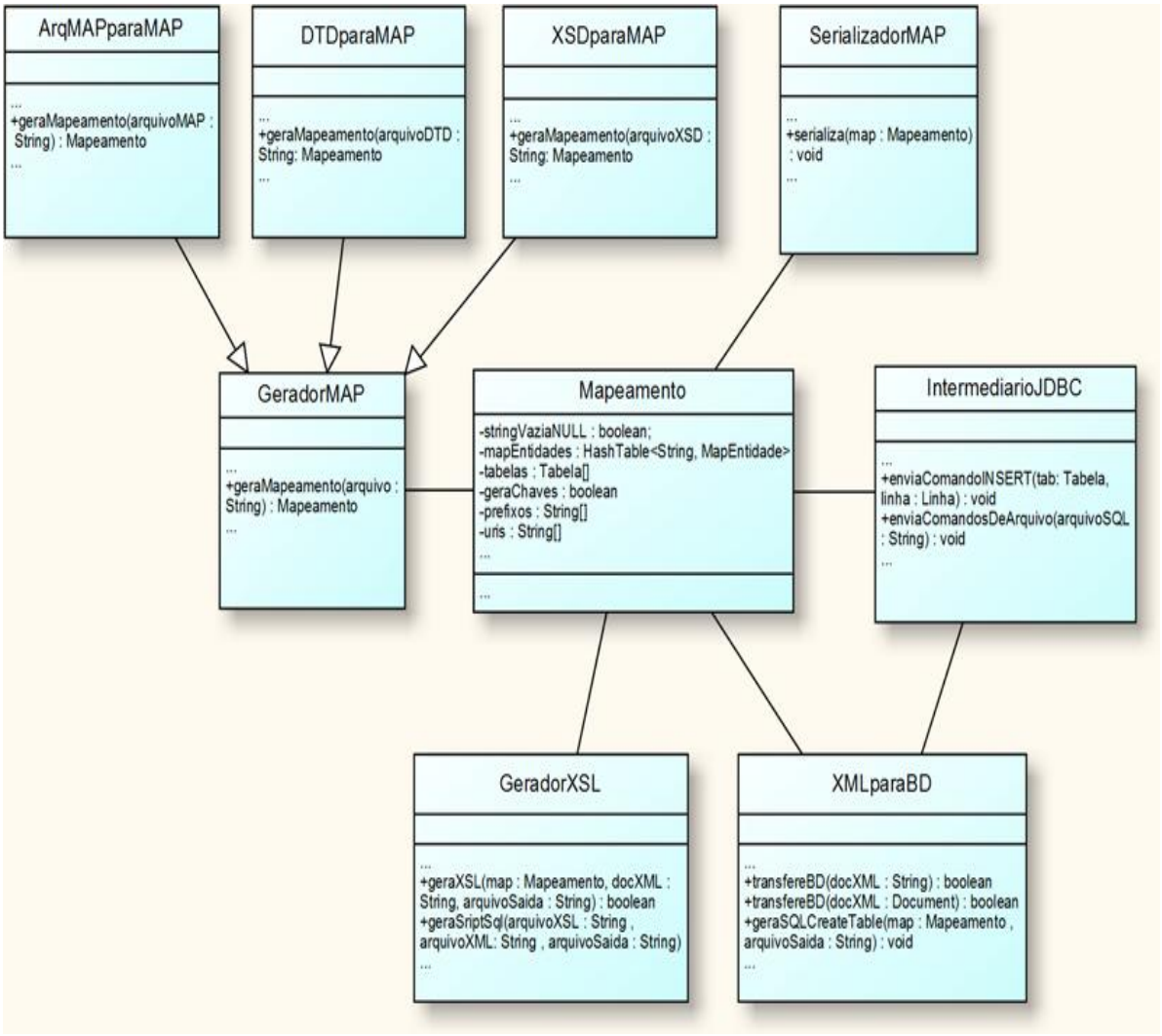

Figura 3: diagrama de classes

A classes ArqMAParaMap, DTDparaMAP e XSDparaMAP estendem a classe GeradorMAP e oferecem entre outras funções, a função geraMapeamento(String arquivo), que retorna um objeto da classe Mapeamento.

A classe SerializadorMAP oferece a função serializa(Mapeamento map, String arquivoSaida), que recebe um objeto Mapeamento e o serializa em forma de um arquivo map.

A classe GeradorXSL oferece a função *geraXSL(Mapeamento map, String saída)* que cria o arquivo de saída que contém comandos XSLT e a função estática *geraScriptSql(String arquivoXSL, String aqruivoXML, String arquivoSaida)* que transforma o documento XML recebido usando os comandos XSLT contidos no arquivo XSL em um script SQL com comandos INSERT.

A classe IntermediarioJDBC oferece funções de acesso ao BD como por exemplo *enviaComandoINSERT(Tabela tab, Linha lin)*, *enviaComandosDeArquivo(String arquivoSQL)*, etc.

A classe XMLparaBD fornece entre outras funções, *transfereBD(String docXML)* e *transfereBD(Document docXML),* que recebem como parâmetro o documento XML e retornam um valor booleano para sucesso ou não. Esta classe oferece ainda a função *getComandosCreateTable(Mapeamento map, String arquivoSaida)*, que recebe um objeto Mapeamento e gera um script SQL com comandos CREATE TABLE.

Foi criada uma classe XmlBDTool que é a ferramenta propriamente dita, que recebe comandos pela linha de comandos e executa uma das funcionalidades do sistema. Por exemplo, para fazer a carga do BD a partir de um documento XML, o comando é:

*java XmlBD*Tool -transf <arquivo_map> <doc_xml> <arquivo_conf>

Note-se que o nome do driver JDBC e a url de conexão com o BD, incluindo nome de usuário e senha, se existir, são especificados no arquivo de configuração em formato texto.

7. Exemplo

Como exemplo, foi escolhido um cenário simples e fictício de uma universidade, cujo BD armazena informações sobre os alunos, professores, turmas, disciplinas e as relações entre os mesmos, conforme mostra o diagrama de entidade-relacionamento da Figura4.

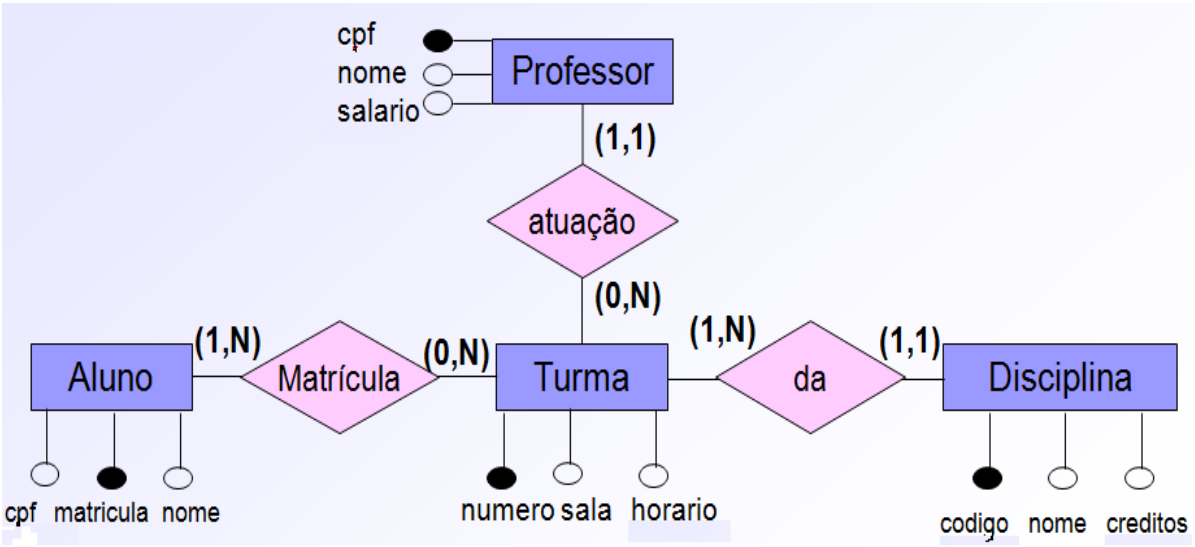

Figura 4: modelo entidade-relacionamento do estudo de caso

Conforme mostrado na figura acima, as entidades envolvidas são: Aluno, Professor, Disciplina e Turma. Entre a entidade Aluno e Turma há um relacionamento de muitos para muitos. Entre as entidades Disciplina e Turma existe um relacionamento de um para muitos, assim como entre as entidades Professor e Turma

O documento XML que contém os dados que serão usados na carga do BD é o seguinte:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Universidade SYSTEM "universidade.dtd">

<Universidade>

  <Aluno matricula="12516LV">
    <cpf>28462887-3</cpf>
    <nome>Miguel Santos</nome>
  </Aluno>
  <Aluno matricula="B23245">
    <cpf>09334734-5</cpf>
    <nome>Maria Cristina Gomes</nome>
  </Aluno>
  <Aluno matricula="3P4235">
    <cpf>98348782-7</cpf>
    <nome>João Delgado</nome>
  </Aluno>
  <Aluno matricula="265365">
    <cpf>28462887-3</cpf>
    <nome>Miguel Santos</nome>
  </Aluno>
  <Aluno matricula="125192">
    <cpf>2842342-3</cpf>
    <nome>Joao Pires Silva</nome>
  </Aluno>


  <Turma numero="98" sala="ctc210">
    <horario>08:10</horario>
    <estudantes>
      <aluno-ref matr="3P4235"/>
      <aluno-ref matr="265365"/>
      <aluno-ref matr="125192"/>
    </estudantes>
    <Professor cpf="8738787-7">
      <nome>Franscisco Pires</nome>
      <salario>5390</salario>
    </Professor>
    <Disciplina codigo="IRF098">
```

```
    <nome>Anatomia I</nome>
    <creditos>5</creditos>
   </Disciplina>
  </Turma>


  <Turma numero="67" sala="ctc112">
   <horario>10:20</horario>
   <estudantes>
    <aluno-ref matr="12516LV"/>
    <aluno-ref matr="B23245"/>
    <aluno-ref matr="125192"/>
    <aluno-ref matr="3P4235"/>
   </estudantes>
   <Professor cpf="232344-9">
    <nome>Manuel Monteiro</nome>
    <salario>4815</salario>
   </Professor>
   <Disciplina codigo="POL87">
    <nome>Nataçao II</nome>
    <creditos>3</creditos>
   </Disciplina>
  </Turma>

</Universidade>
```

Foram criadas no BD as tabelas TAluno, TProfessor, TDisciplina, TTurma e a tabela TMatricula que é a tabela associativa entre TAluno e TTurma.

O arquivo de mapeamento XML-Relacional é apresentado a seguir:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE XMLparaBD SYSTEM "C:\Users\Edwaldo\tcc\XMLparaBD.dtd">
<XMLparaBD Versao="1.0">
 <MapXMLRel>

  <!-- Ignora o elemento Universidade e mapeia Aluno e Turma -->
  <IgnoraElementoRaiz>
   <Elemento Nome="Universidade"/>
   <FilhoRaiz>
    <Elemento Nome="Aluno"/>
   </FilhoRaiz>
   <FilhoRaiz>
    <Elemento Nome="Turma"/>
   </FilhoRaiz>
  </IgnoraElementoRaiz>
```

```
<!-- Mapeamento de Aluno -->
<Entidade>
  <Elemento Nome="Aluno"/>
  <Tabela Nome="TAluno"/>
  <Propriedade>
    <Atributo Nome="matricula"/>
    <Coluna Nome="matricula"/>
  </Propriedade>
  <Propriedade>
    <Elemento Nome="nome"/>
    <Coluna Nome="nome"/>
  </Propriedade>
  <Propriedade>
    <Elemento Nome="cpf"/>
    <Coluna Nome="cpf"/>
  </Propriedade>
</Entidade>

<!-- Mapeamento de Turma -->
<Entidade>
  <Elemento Nome="Turma"/>
  <Tabela Nome="TTurma"/>
  <Propriedade>
    <Atributo Nome="numero"/>
    <Coluna Nome="numero"/>
  </Propriedade>
  <Propriedade>
    <Atributo Nome="sala"/>
    <Coluna Nome="sala"/>
  </Propriedade>
  <Propriedade>
    <Elemento Nome="horario"/>
    <Coluna Nome="horario"/>
  </Propriedade>


<!-- Mapeamento do relacionamento N-N entre Aluno e Turma -->
  <EntidadeRelacionada tipoRelacionamento="N-N">
    <Elemento nome="Aluno"/>
    <TabelaAssociativa nome="TMatricula" elemento="estudantes/aluno-ref" idref="matr">
      <Coluna nome="matr_aluno" valor="TAluno.matricula"/>
      <Coluna nome="num_turma" valor="TTurma.numero"/>
    </TabelaAssociativa>
  </EntidadeRelacionada>

  <!-- Mapeamento do relacionamento 1-N entre Professor e Turma-->
  <EntidadeRelacionada tipoRelacionamento="Direto">
    <Elemento nome="Professor"/>
    <ChavesJuncao ChaveEntidadePai="Estrangeira">
      <ChaveUnica gerarAutomaticamente="N">
        <Coluna nome ="cpf"/>
```

```
      </ChaveUnica>
      <ChaveEstrangeira>
        <Coluna nome ="cpf_professor"/>
      </ChaveEstrangeira>
    </ChavesJuncao>
  </EntidadeRelacionada>

  <!-- Mapeamento do relacionamento 1-N entr Disciplina e Turma-->
  <EntidadeRelacionada tipoRelacionamento="Direto">
    <Elemento nome="Disciplina"/>
    <ChavesJuncao ChaveEntidadePai="Estrangeira">
      <ChaveUnica gerarAutomaticamente="N">
        <Coluna nome ="codigo"/>
      </ChaveUnica>
      <ChaveEstrangeira>
        <Coluna nome ="cod_disciplina"/>
      </ChaveEstrangeira>
    </ChavesJuncao>
  </EntidadeRelacionada>

</Entidade>

<!-- Mapeamento de Professor -->
<Entidade>
  <Elemento Nome="Professor"/>
  <Tabela Nome="TProfessor"/>
  <Propriedade>
    <Atributo Nome="cpf"/>
    <Coluna Nome="cpf"/>
  </Propriedade>
  <Propriedade>
    <Elemento Nome="nome"/>
    <Coluna Nome="nome"/>
  </Propriedade>
  <Propriedade>
    <Elemento Nome="salario"/>
    <Coluna Nome="salario"/>
  </Propriedade>
</Entidade>

<!-- Mapeamento de Disciplina -->
<Entidade>
  <Elemento Nome="Disciplina"/>
  <Tabela Nome="TDisciplina"/>
  <Propriedade>
    <Atributo Nome="codigo"/>
    <Coluna Nome="codigo"/>
  </Propriedade>
  <Propriedade>
    <Elemento Nome="nome"/>
    <Coluna Nome="nome"/>
```

```
    </Propriedade>
    <Propriedade>
      <Elemento Nome="creditos"/>
      <Coluna Nome="creditos"/>
    </Propriedade>
  </Entidade>

 </MapXMLRel>
</XMLparaBD
```

7. Conclusão

Este trabalho teve como objetivo o desenvolvimento de uma ferramenta de carga de BD relacionais para fontes de dados XML.

Foi desenvolvido um conjunto de classes e interfaces (API) que permitem que o sistema desenvolvido possa ser chamado por diferentes aplicações. Isto significa que a ferramenta pode ser utilizada diretamente pelo usuário ou, então, em outras aplicações através da API desenvolvida.

Devido à importância e popularidade cada vez maior do XML como padrão na integração de sistemas, muito se tem pesquisado e desenvolvido para facilitar a carga de BDs relacionais a partir de fontes de dados XML. Porém, a grande maioria das soluções disponíveis no mercado é de código fechado e licença comercial. Outras são soluções complexas e difíceis de usar, ou ainda presas a determinados formatos e dependentes de determinados SGBD.

Neste sentido, foi desenvolvido o presente trabalho, que apresenta uma possível solução simples, multi-plataforma e independente de SGBD, para as necessidades de carga de bancos relacionais a partir de dados contidos em documentos XML.

A ferramenta desenvolvida apresenta alguns diferenciais em relação às demais existentes de código aberto. Além de fornecer facilidades para a geração de arquivos de mapeamento, como a geração automática a partir de DTD e XML Schema, ela consegue gerar scripts XSLT que quando executados sobre os documentos XML com os dados, os transforma em scripts SQL com os comandos INSERT necessários. Tais facilidades não foram encontradas em nenhuma das ferramentas de código aberto estudadas no capítulo de trabalhos relacionados.

Referências

W3C; **Extensible Markup Language (XML)**, 2011. Disponível em: < http://www.w3.org/XML/>. Acesso em 12 de Fevereiro de 2011

BEZA, O; PATSALA, M; KERAMOPOULUS, E. **Comparison of XML Support in IBM DB2, Microsoft SQL Server, Oracle**, Dpt. Of Information Technology, Thessaloniki, Greece, 2008

BOURRET, R; **XML Database products,** 2010. Disponível em: < http://www.rpbourret.com/xml/XMLDatabaseProds.htm#integration >. Acesso em 5 de fevereiro de 2011.

ALTOVA. **MapForce – Graphical Data Mapping, Conversion, and Integration Tool**. Disponível em: <http://www.altova.com/mapforce.html>. Acesso em 21 de fevereiro de 2011

EXOLAB.ORG. **The Castor Project**. Disponível em < http://www.castor.org/ >. Acesso em 11 de fevereiro de 2011

ETASOFT. **Extreme Translator**. Disponível em: < http://www.xtranslator.com/et.htm>. Acesso em: 17 de fevereiro de 2011

HIT SOFWARE. **Allora, Graphical DatabaseXML Mapping and Transformation**. Disponível em: < http://www.hitsw.com/products_services/xmlplatform.html>. Acesso em 22 de fevereiro de 2011

IBM**. DB/XML Transform™.** Disponível em: < http://www.treehouse.com/DBXMLTransform.shtml>. Acesso em 17 de fevereiro de 2011

MICROSOFT. **Persisting Records in XML Format**. Disponível em: < http://msdn.microsoft.com/en-us/library/ms681538.aspx>. Acesso em: 10 de fevereiro de 2011

MICROSOFT. **SQLXML**. Disponível em: < http://msdn.microsoft.com/en-us/library/aa286527.aspx>. Acesso em: 5 de fevereiro de 2011

ODONATA**. XQuare Bridge: extending relational databases with XQuery and XML-relational mapping D**isponível em: < http://xquare.ow2.org/bridge/index.html>. Acesso em 11 de fevereiro de 2011

ORACLE**. Oracle XML Developer's Kit (XDK) Java - 10.2.0.2.0 LINUX Production**. Disponível em: < http://www.oracle.com/technetwork/database/features/xmldb/java-utilsoft-087831.html>. Acesso em 12 de fevereiro de 2011.

SOFTRUS. **Db To Xml**. Disponível em: < http://www.soft-r-us.com/dbtoxml.asp > Acesso em: 16 de fevereiro de 2011

STEWART, G; **Osage - Persistence Plus XML**. Disponível em: < http://osage.sourceforge.net/>. Acesso em12 de fevereiro de 2011.