UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA
GRUPO DE LINGUAGENS DE PROGRAMAÇÃO E BANCO DE DADOS

PROPAGAÇÃO DE MODIFICAÇÕES EM DOCUMENTOS XML PARA BANCOS DE DADOS RELACIONAIS

DISSERTAÇÃO DE MESTRADO

Regis Rodolfo Schuch

Santa Maria, RS, Brasil

2014

PROPAGAÇÃO DE MODIFICAÇÕES EM DOCUMENTOS XML PARA BANCOS DE DADOS RELACIONAIS

Regis Rodolfo Schuch

Dissertação apresentada ao Programa de Pós-Graduação em Informática (PPGI),
da Universidade Federal de Santa Maria (UFSM, RS),
como requisito parcial para obtenção do grau de
Mestre em Informática

Orientadora: Profª. Drª. Deise de Brum Saccol

Santa Maria, RS, Brasil

2014

Rodolfo Schuch, Regis

PROPAGAÇÃO DE MODIFICAÇÕES EM DOCUMENTOS XML PARA BANCOS DE DADOS RELACIONAIS / por Regis Rodolfo Schuch. – 2014.

122 f.: il.; 30 cm.

Orientadora: Deise de Brum Saccol

Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2014.

1. XML. 2. Visões. 3. Atualizações. I. Saccol, Deise de Brum. II. Título.

E-mail: regis.schuch@gmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

PROPAGAÇÃO DE MODIFICAÇÕES EM DOCUMENTOS XML PARA BANCOS DE DADOS RELACIONAIS

elaborada por
Regis Rodolfo Schuch

Como requisito parcial para obtenção do grau de
Mestre em Informática

COMISSÃO EXAMINADORA:

Deise de Brum Saccol, Drª.
(Presidente/Orientadora)

Eduardo Nunes Borges, Dr. (FURG)

Sergio Luis Sardi Mergen, Dr. (UFSM)

Santa Maria, 09 de maio de 2014.

AGRADECIMENTOS

Neste trabalho de mestrado, tive oportunidade de constatar que uma dissertação, apesar do processo solitário a que qualquer pesquisador está destinado, reúne contribuições de várias pessoas e instituições, as quais foram fundamentais para a conclusão deste trabalho.

À Professora Deise de Brum Saccol, minha orientadora e exemplo profissional, agradeço por não ter permitido que eu interrompesse o processo e pela confiança. Acima de tudo, obrigado por me acompanhar nesta jornada e pelas valiosas contribuições para o trabalho.

Aos professores e colegas do Curso de Pós-Graduação em Informática da UFSM. Aos professores Giovani Rubert Librelotto e Juliana Kaizer Vizzotto, que aceitaram compor minha banca de seminário de andamento, pelas sugestões e análises significativas às quais tentei atender na versão final da dissertação.

À Universidade Federal de Santa Maria, pela possibilidade de uma ótima formação. Pelo apoio financeiro, agradeço ao órgão do governo federal CAPES.

Sou muito grato a todos os meus familiares. A minha mãe, pelo incentivo e por ter oportunizado o acesso ao Ensino superior, sei o esforço que isto representou, serei eternamente grato. A minha irmã, grande companheira durante esta caminhada, pelo apoio e amizade.

Aos amigos pelo apoio e pelo que representam em minha vida.

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

PROPAGAÇÃO DE MODIFICAÇÕES EM DOCUMENTOS XML PARA BANCOS DE DADOS RELACIONAIS

AUTOR: REGIS RODOLFO SCHUCH
ORIENTADORA: DEISE DE BRUM SACCOL
Data e Local da Defesa: Santa Maria, 9 de maio de 2014.

Muitos ambientes requerem o armazenamento de dados XML (*eXtensible Markup Language*) em BDR (Bancos de Dados Relacionais). Nesses casos, além da mapear o esquema e os dados XML para tabelas no BDR, é necessária a atualização do BDR na medida em que os dados XML sofrem modificações. Para realizar a atualização, este trabalho propõe o DBUpdater (*Database Updater*), um componente que detecta e propaga modificações de dados XML para BDR. A propagação de modificações no conteúdo dos documentos XML para o BDR não é uma tarefa cuja execução é direta. Devida à ausência de um meio para rastrear os dados XML, é necessário um mecanismo para gerar chaves XML que permitem que os dados XML se relacionem com as tuplas nas tabelas relacionais. As principais contribuições deste trabalho são: a detecção de modificações com base em algoritmos de *Diff*; a geração e atribuição de identificadores para os nodos XML; a geração de chaves para os dados XML com base nos identificadores atribuídos aos nodos; e a definição de equivalência entre as operações de modificação detectadas em relação às operações de atualização do banco de dados. Esta proposta está inserida no *framework* X2Rel (*XML to Relational*), um ambiente de armazenamento, atualização e consulta a dados XML heterogêneos em BDR.

Palavras-chave: XML, visões e atualização.

ABSTRACT

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

CHANGE PROPAGATION FROM XML DOCUMENTS TO RELATIONAL DATABASES

AUTOR: REGIS RODOLFO SCHUCH
ORIENTADORA: DEISE DE BRUM SACCOL
Data e Local da Defesa: Santa Maria, 9 de maio de 2014.

Many environments require the storage of XML data (eXtensible Markup Language) in RDB (Relational Databases). In such cases, in addition to mapping the XML schema and data to tables in the RDB, it is also necessary to update the RDB once the XML data are modified. To perform the updating process, this work proposes the DBUpdater (Database Updater), a component that detects and propagates XML changes to a RDB. The content change propagation from XML documents to the RDB is not a task whose execution is straightforward. Due to the absence of a means to track the XML data, a mechanism is needed to generate XML keys which allows the XML data relate to the tuples in the relational tables. The main contributions of this work are: the change detection based on diff algorithms; the generation and assignment of identifiers for the XML nodes; the generation of keys for the XML data based on the identifiers assigned to the nodes; and the definition of equivalence between the detected change operations in relation to the database update operations. This proposal is part of X2Rel (XML to Relational) framework, an environment for storing, maintaining and querying heterogeneous XML data in RDB.

Keywords: XML, views and updating.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

BDR	Banco de Dados Relacionais
CMap	*Concept Mapper*
DBLP	*Digital Bibliography & Library Project*
DBUpdater	*Database Updater*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
DOM	*Document Object Model*
DTD	*Document Type Definition*
ER	*International Conference on Conceptual Modeling*
GPLPBD	Grupo de Pesquisas em Linguagens de Programação e Banco de Dados
IdMapper	*Identifier Mapper*
MDL	*Multi Dimension Link*
OntoGen	*Ontology Generator*
OntoRel	*Ontology to Relational*
OWL	*Web Ontology Language*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
UpMapper	*Update Mapper*
VLDB	*Very Large Data Bases*
X2Rel	*XML to Relational*
XKGen	*XML Key Generator*
XMap	*XML Mapping*
XML	*eXtensible Markup Language*
XQuery	*XML Query Language*

SUMÁRIO

1. INTRODUÇÃO

A linguagem XML vem se destacando como um importante padrão para a representação e compartilhamento de dados. Esse sucesso é determinado, em grande parte, pela sua capacidade em representar dados semiestruturados, gerados pelas mais diversas aplicações. Frente ao crescimento na utilização da XML, tem surgido demanda por novas alternativas de armazenamento de dados nesse formato.

Atualmente, os dados XML podem ser armazenados em sistemas de arquivos, em SGBD (Sistema Gerenciador de Banco de Dados) Relacional/Objeto Relacional e em bancos de dados XML nativos. Nos SGBD, basicamente, pode-se optar pelo armazenamento dos dados como conteúdo de coluna e como tabelas. A primeira alternativa utiliza banco de dados com extensão à XML, como os campos do tipo XMLType (ORACLE, 2002). Já a segunda é feita através do mapeamento entre as estruturas de dados. Entretanto, é importante compreender a quais problemas cada forma de armazenamento se aplica melhor, pois isso torna possível escolher a alternativa mais apropriada para cada situação.

Em ambientes onde a presença de heterogeneidade na representação dos dados XML é uma realidade, abordagens estão sendo apresentadas com o objetivo de gerenciar essas diversidades estruturais. Exemplos de abordagens são aquelas usadas para armazenar dados XML em BDR, como encontrado em (VYSNIAUSKAS; NEMURAITE, 2006), (ZHANG et al. 2009) e (SACCOL; ANDRADE; PIVETA, 2011). Tais abordagens costumam definir um esquema relacional a partir das estruturas XML, conforme exemplo ilustrado através da Figura 1.1, e, em seguida, inserir os dados XML como tuplas do BDR. Se uma atualização for efetuada nos dados XML, esta operação deve ser propagada para o BDR.

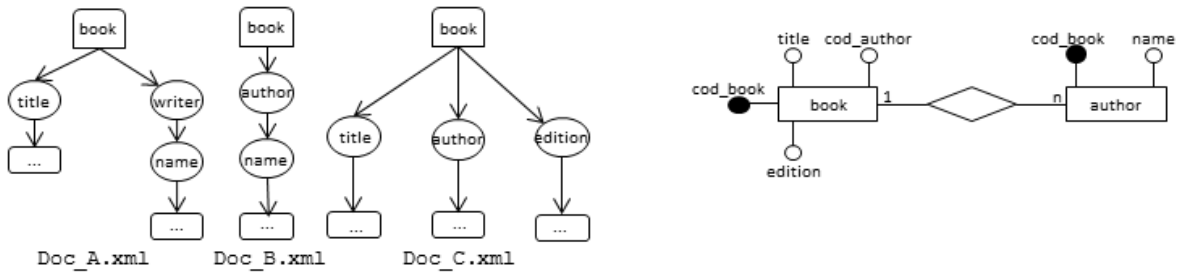


Figura 1.1. Esquema relacional definido a partir de documentos XML.

Duas questões merecem especial atenção para propagar as modificações dos dados XML para o BDR: (i) a detecção das modificações; e (ii) o rastreamento dos dados XML com as tuplas das tabelas do BDR. Para solucionar o primeiro problema, estão disponíveis alguns algoritmos para detecção de modificações (algoritmos de *Diff*), como encontrado em (COBÉNA; ABITEBOUL; MARIAN, 2002), (WANG; DEWITT; CAI, 2003) e (AL-EKRAM; ADMA; BAYSAL, 2005). No entanto, definir o algoritmo adequado ao contexto de aplicação ainda representa um desafio na área de pesquisa. Algoritmos de *Diff* para este propósito devem considerar, além das modificações de conteúdo e de estrutura, outras características relevantes para a aplicação em questão, tais como o modelo de ordenação da árvore XML, o modelo de operações e o formato do arquivo de saída, também chamado de delta. Analisando a Figura 1.2, há duas versões do mesmo documento XML (`Doc_A.xml`), sendo, `Old.xml` e `New.xml`, bem como o delta gerado a partir da comparação entre as versões.

Documento Old.xml (fragmentado)

```
 ...
 3.<book year="1999">
 4. <title>The Economics of...</title>
 5. <author>
 6.  <names>
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10. </author>
11. <publisher>Springer</publisher>
12. <price>129.00</price>
13.</book>...
```

Delta

```
10 insert
 <instituition>CITI</instituition>
13 insert
 <edition>1999 edition</edition>
14 update
 <price>223.48</price>
```

Documento New.xml (fragmentado)

```
 ...
 3.<book year="1999">
 4. <title>The Economics of...</title>
 5. <author>
 6.  <names>
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10.  <instituition>CITI</instituition>
11. </author>
12. <publisher>Springer</publisher>
13. <edition>1999 edition</edition>
14. <price>223.48</price>
15.</book>...
```

Figura 1.2. Representação das versões do documento XML e do delta.

O delta diz que o conteúdo da linha `12` do documento `Old.xml` foi editado. Ou seja, o preço do livro foi modificado para o valor `223.48`, conforme linha `14` do documento `New.xml`. Do mesmo modo, o delta diz que foi inserida a instituição na qual o autor está afiliado (linha `10`), bem como a edição do `book` (linha `13`). Considerando uma estrutura de BDR definida a partir de documentos XML, uma sequência de comandos (*insert*, *delete* ou *update*) em

linguagem SQL (*Structured Query Language)* para executar no banco de dados podem ser gerados a partir do delta.

A propagação de modificações coloca um segundo desafio, que é o de estabelecer uma estratégia para rastrear os dados XML com as tuplas. Esta dificuldade ocorre na ausência de chaves que permitam referenciar uma entidade à outra. A Figura 1.3 exemplifica o esquema de tabelas gerado a partir das versões do documento XML ilustrado na Figura 1.2. O conteúdo do documento (ou seja, os dados de texto) é mapeado e inserido nas respectivas tabelas. Como não existe uma chave pré-definida, os valores são gerados e atribuídos de maneira aleatória.

book

cod book	title	year	publisher	edition	price	cod author
20	The Economics of...	1999	Springer	1999 edition	223.48	1
...	...	...	...	...	...	...

author

cod author	instituition
1	CITI
...	...

author_names

cod_author	cod_names
1	7
...	...

names

cod names	first	last
7	Darcy	Gerbarg
...	...	...

Figura 1.3. Tabelas relacionais geradas a partir de documentos XML.

Entretanto, em bancos de dados que armazenam dados XML, o cenário ideal é quando as tuplas compartilham a mesma chave dos dados XML; assim, tais chaves fornecem uma condição para rastrear os dados dos documentos XML com os dados do BDR. No entanto, os documentos XML podem não estar estruturados com chaves, como os documentos exemplificados na Figura 1.2, tornando o processo de propagação das modificações para o BDR mais difícil. As soluções já existentes para propagação de modificações limitam-se ao rastreamento de chaves definidas com base na estrutura individual dos documentos XML, o que não se aplica ao contexto em que há diversidade estrutural dos documentos.

Para lidar com essa questão das chaves, este trabalho propõe gerar chaves para os dados XML de acordo com o conceito dos nodos no esquema relacional, ou seja, se corresponde à tabela ou coluna. A geração dessas chaves pode ser feita a partir do rótulo identificador do nodo, cuja finalidade é identificar a absoluta posição de cada nodo na estrutura XML (TATARINOV et al., 2002). Nesse cenário, o rastreamento dos dados XML com as tuplas no BDR torna-se mais simples.

Para transpor os dois problemas discutidos nesta seção, este trabalho apresenta a abordagem DBUpdater (*Database Updater*), que permite identificar e propagar modificações de dados XML para operações (instruções SQL para inserir, excluir ou atualizar) de

modificação no BDR. Essa abordagem faz parte do *framework* X2Rel (SACCOL; ANDRADE; PIVETA, 2011), um ambiente de armazenamento, atualização e consulta a dados XML heterogêneos armazenados em BDR. Entretanto, se ocorrer modificação na estrutura dos documentos XML, a devida propagação para o esquema relacional é assumida por outro trabalho, ainda em desenvolvimento.

1.1. Objetivos e Contribuições

O objetivo geral deste trabalho é propor uma abordagem que permita identificar e propagar modificações do conteúdo de documentos XML para instruções de atualização do BDR. Devido à ausência de meios para rastrear os dados XML, a propagação das modificações não é uma tarefa cuja execução seja direta. Diante disso, propõe-se um mecanismo gerador de chaves XML que torna possível relacionar os dados XML com as tuplas nas tabelas relacionais.

Os principais objetivos específicos e contribuições do trabalho são:

- A criação e atribuição de identificadores para os nodos XML, preservando a identidade do nodo, mesmo após modificações na estrutura do documento;
- A geração de chaves para os dados XML com base nos identificadores atribuídos aos nodos e no seu conceito no esquema relacional;
- A definição de equivalência entre as operações de modificação detectadas nos documentos XML em relação às operações de atualização sobre o BDR, na linguagem SQL.

1.2. Organização do Texto

A escrita da dissertação está estruturada como segue:

- O Capítulo 2 – **Fundamentação Teórica** – apresenta as pesquisas existentes que estão relacionadas aos problemas específicos tratados abordagem proposta para o DBUpdater. Além disso, são apresentados os trabalhos relacionados à manutenção de visões;

- O Capítulo 3 – **Proposta do Trabalho** – propõe a arquitetura da abordagem proposta neste trabalho, além da geração de chaves para rastrear os dados XML com as tuplas relacionais e da identificação de equivalência entre às operações XML com SQL;
- O Capítulo 4 – **Experimentos e Avaliação de Resultados** – demonstra a aplicabilidade do trabalho e avalia a abordagem proposta através da análise dos resultados obtidos;
- O Capítulo 5 – **Conclusões** – apresenta as principais conclusões do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Neste trabalho, a detecção de diferenças e o uso de chaves são as abordagens adotadas para identificar e propagar modificações em documentos XML para o BDR. O problema tratado na dissertação é parte do *framework* X2Rel (SACCOL; ANDRADE; PIVETA, 2011), um ambiente para mapeamento de estrutura, dados, atualizações e consultas em documentos XML com incompatibilidades estruturais e semânticas para BDR. Conceitos e tecnologias recorrentes são apresentados ao longo deste capítulo, além dos trabalhos relacionados ao tema da pesquisa.

2.1. *Framework* X2Rel

Em cenários onde as aplicações requerem a materialização de visões relacionais de dados de arquivos XML, a presença de heterogeneidade na estrutura dos arquivos é uma realidade. Neste sentido, a proposta apresentada pelo *framework* X2Rel pode ser de grande importância, principalmente pelo processo de unificação das estruturas de documentos XML e mapeamento de equivalências entre o modelo XML e relacional.

O *framework* X2Rel está inserido em um ambiente para armazenamento de dados XML em BDR. Sua abordagem usa ontologias para mapear a estrutura XML para um esquema relacional. Dessa forma, documentos XML com estruturas diferentes podem ser adequadamente armazenados em um BDR, o qual se considera uma visão materializada dos dados XML. Para permitir o mapeamento completo – de estrutura, dados, atualização e consultas – o X2Rel apresenta as seguintes funcionalidades:

- OntoGen (*Ontology Generator*): padroniza a estrutura dos documentos XML em um esquema global descrito por uma ontologia. Este componente recebe um conjunto de documentos XML e produz a ontologia (esquema integrado) (MELLO, 2007) (SACCOL et al., 2008);
- OntoRel (*Ontology to Relational*): traduz a ontologia para um esquema relacional. Este componente recebe a ontologia e produz um esquema relacional (um *script* SQL com instruções DDL) (SACCOL; ANDRADE; PIVETA, 2011);

- CMap (*Concept Mapper*): descreve as equivalências de conceitos entre os documentos XML, a ontologia e o esquema relacional. Este componente recebe os documentos XML, a ontologia e o esquema relacional, e produz o documento de mapeamento (um documento XML com as equivalências detectadas) (NEGRINI et al., 2012);
- XMap (*XML Mapper*): mapeia e insere os dados XML no BDR. Este componente recebe os documentos XML, a ontologia e o esquema relacional, e produz o *script* com um conjunto de instruções SQL DML de inserção (AVELAR; SACCOL, PIVETA, 2012);
- SUpdater (*Schema Updater*): detecta e propaga modificações do esquema dos documentos XML para o esquema relacional. Este componente recebe os documentos XML, a ontologia e o esquema relacional, e produz um *script* com um conjunto de instruções SQL DDL para alteração da estrutura do banco de dados;
- QMap (*Query Mapper*): traduz as consultas XML originais em instruções SQL equivalentes. Este componente recebe uma consulta XQuery (*XML Query Language*) e produz a consulta SQL correspondente (LOOSE; SACCOL, 2012);

A arquitetura do *framework* X2Rel pode ser observada na Figura 2.1.

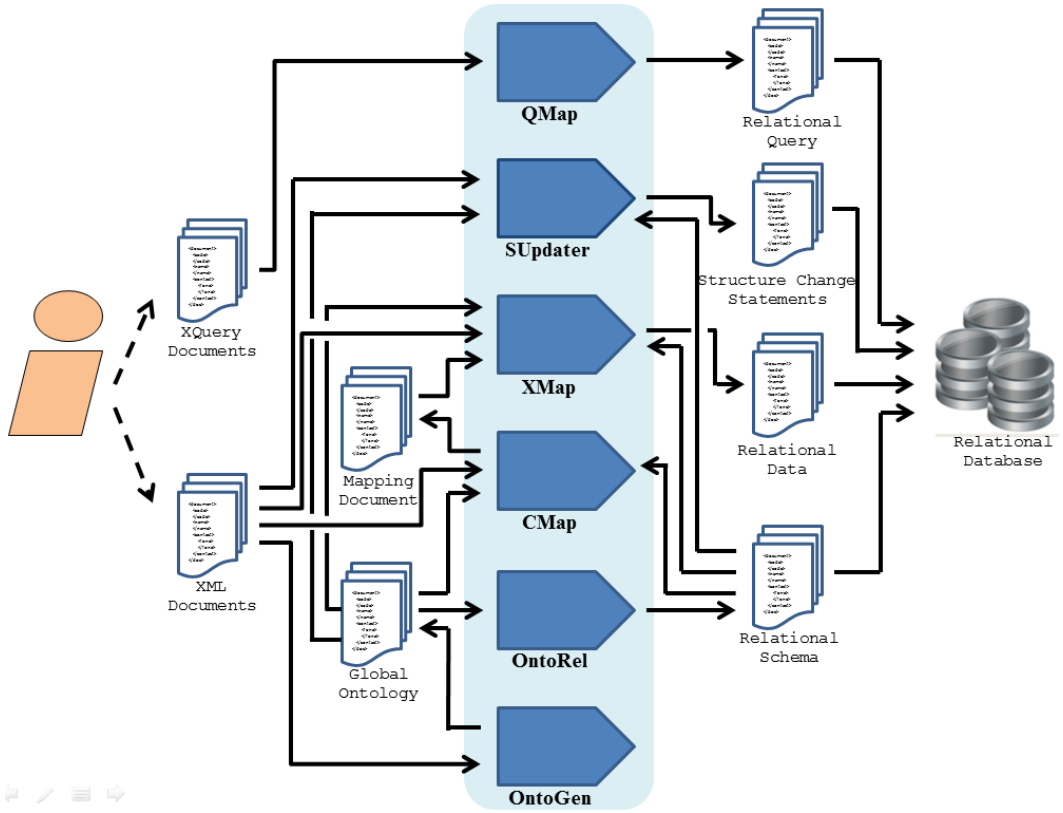


Figura 2.1. Arquitetura do *framework* X2Rel.

Somente o componente SUpdater é um projeto em andamento. Os demais componentes – OntoGen, OntoRel, CMap, XMap e QMap (desenvolvidos no âmbito grupo de pesquisa GPLPBD (Grupo de Pesquisas em Linguagens de Programação e Banco de Dados) do PPGI da UFSM) – são trabalhos finalizados e publicados.

O processo de atualização do banco de dados delimita o escopo do presente trabalho no contexto do X2Rel. Especificamente, a proposta destina-se à abordagem do componente DBUpdater (*Database Updater*), o qual é responsável por propagar as modificações nos dados XML para instruções de atualização do BDR, através do rastreamento de dados a partir das chaves.

A abordagem para geração das chaves também é proposta neste trabalho, através do componente XKGen (*XML Key Generator*), e externalizada na arquitetura do X2Rel para ser usada, também, na etapa inicial de armazenamento dos dados XML no BDR, realizada por XMap. Dessa forma, o valor da chave primária, antes gerado aleatoriamente por XMap, agora passa a ser definido com base nos chave XML gerada por XKGen.

Os componentes DBUpdater e XKGen são apresentados de maneira detalhada no capítulo 3. A seguir, os componentes OntoRel e CMap são detalhados, sendo que os mesmos são utilizados no processo de geração das chaves e composição das instruções de atualização. O componente XMap é descrito de forma geral na seção 2.1.3, sendo que este é utilizado pelo DBUpdater para gerar a operação de atualização *insert*.

2.1.1. Componente OntoRel

O componente OntoRel é responsável por criar o esquema lógico relacional a partir de uma ontologia (SACCOL; ANDRADE; PIVETA, 2011). Para isso, recebe como entrada uma ontologia descrita na linguagem OWL (*Web Ontology Language*), gerada pelo componente OntoGen, e a traduz para um esquema de banco de dados. O processo de tradução é dividido em etapas, conforme ilustrado na Figura 2.2.

Inicialmente, a ontologia é carregada e os objetos do tipo Conceito não-léxico, Conceito léxico ou Relacionamento são gerados.

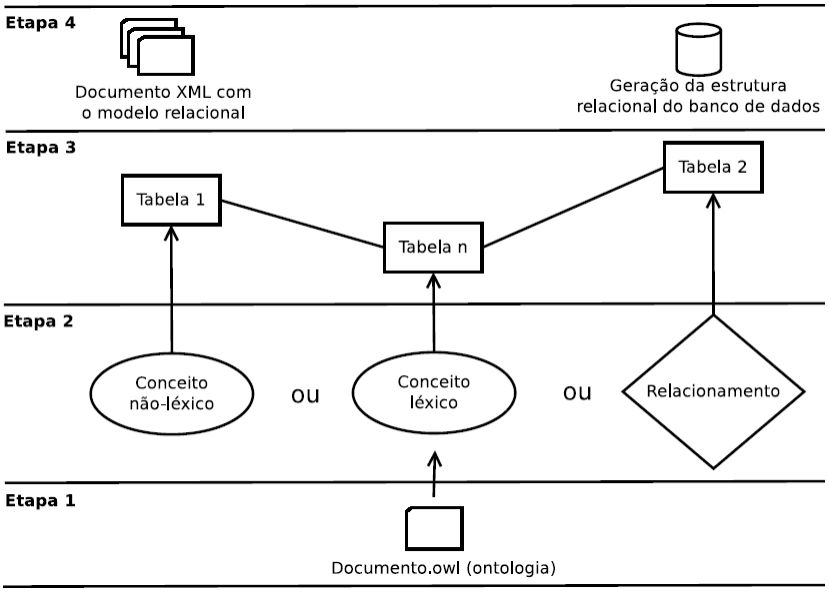


Figura 2.2. Arquitetura do componente OntoRel (SACCOL; ANDRADE; PIVETA, 2011).

O processo para tradução dos conceitos da ontologia para o esquema lógico relacional segue às seguintes regras:

- Conceito não-léxico[1]: é mapeado para uma tabela do esquema relacional. Tal tabela recebe o nome do conceito e é criada uma chave primária com o nome `cod_` mais o nome da tabela.
- Conceito léxico[2]: é mapeado para uma coluna da tabela correspondente ao conceito não-léxico a que pertence. Os relacionamentos que associam um conceito não-léxico e um conceito-léxico determinam que a tabela gerada para o conceito não-léxico recebe uma coluna com o nome e tipo do conceito léxico associado.
- Relacionamento: corresponde às cardinalidades direta e inversa entre os conceitos da ontologia. De acordo com a cardinalidade são determinadas as chaves primárias e estrangeiras, além da obrigatoriedade (*not null*) ou não (*null*) dos campos.

Após aplicadas as regras de transformação, armazenam-se as informações em objetos do tipo tabela. Na quarta e última etapa, é construído o documento XML contendo o esquema das tabelas e o *script* SQL que pode ser salvo em um arquivo *.sql* ou executado diretamente no banco de dados.

[1] Consiste em informações que não podem ser representadas diretamente em um computador (objetos complexos).
[2] Caracterizado por armazenar informações que podem ser representadas diretamente em um computador através de cadeias de bits (inteiros, cadeias de caracteres, etc.)

No presente trabalho, o esquema lógico relacional no formato de documento XML é de interesse para identificar o atributo chave da correspondente tabela. Desse modo, no Apêndice C é ilustrado um exemplo do esquema que é usado no decorrer desta dissertação.

2.1.2. Componente CMap

O componente CMap (NEGRINI et al., 2012) é responsável pela geração e representação das equivalências entre os arquivos XML de entrada, a ontologia global (gerada por OntoGen) e o esquema relacional (gerado por OntoRel). A geração das equivalências é feita a partir dos conceitos e relacionamentos presentes na ontologia, e de um conjunto de assertivas de correspondências para resolução de conflitos. Como artefato de saída, as equivalências são documentadas em um documento de mapeamento descrito na linguagem XML.

O processo de geração e documentação das equivalências está dividido em quatro níveis, como ilustrado na Figura 2.3. No primeiro nível são carregados os documentos ou esquemas XML, a ontologia global e o esquema relacional fornecidos como entrada. O processo é otimizado pela entrada dos esquemas, pois apenas a estrutura do documento XML é descrita. Também é possível usar documentos XML diretamente como entrada, porém, o processo torna-se mais lento, pois é necessário extrair o esquema a partir da estrutura do arquivo. A extração do esquema XML é feita no segundo nível.

O processo de interpretar os esquemas XML, a ontologia e a descrição do esquema relacional, de forma que os conceitos e relacionamentos sejam representados em memória de forma adequada, é realizado no terceiro nível. A partir disso, são geradas as equivalências entre os artefatos de entrada.

O quarto e último nível tem a função de documentar as equivalências detectadas no terceiro nível. Como artefato de saída, é gerado um documento de mapeamento na linguagem XML. A partir do documento de mapeamento, é possível partir de um conceito de um documento XML e encontrar seu equivalente no modelo relacional, bem como saber qual conceito de cada um dos arquivos XML está relacionado em uma coluna ou tabela do banco de dados.

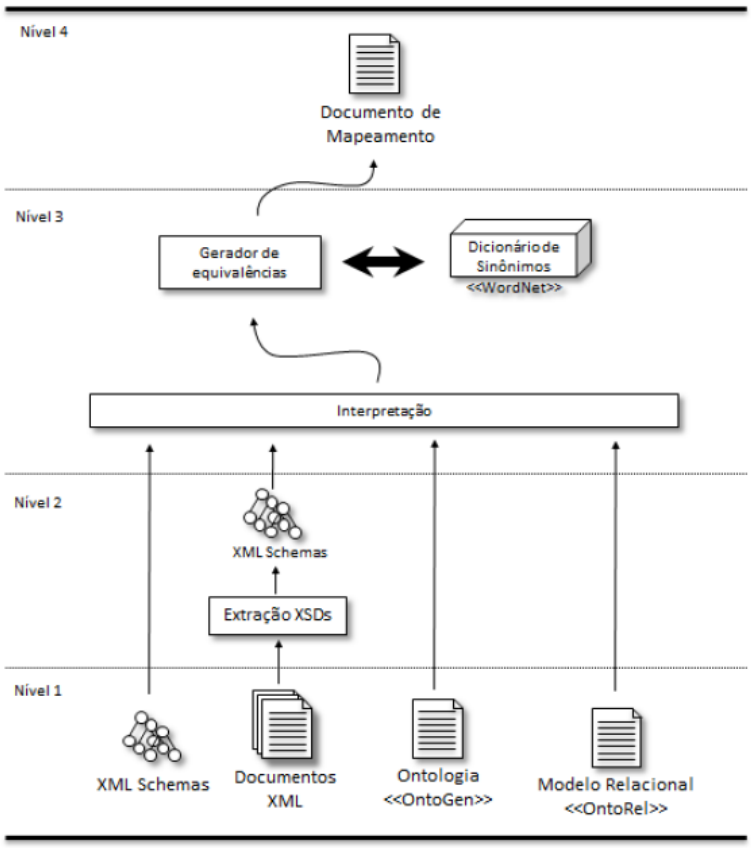


Figura 2.3. Arquitetura do componente CMap (NEGRINI et al., 2012).

Em razão do DBUpdater usar o documento de mapeamento para identificar a equivalência das entidades XML no esquema relacional, um exemplo prático é apresentado para a compreensão do processo de representação das equivalências.

Conforme ilustrado na Figura 2.4, verifica-se que `Doc_A.xml` possui o elemento `<author>` (linha 5), assim como `Doc_B.xml` possui o elemento `<writer>` (linha 5), ambos equivalentes ao conceito `author` na ontologia representativa. A ontologia é traduzida para o esquema lógico relacional, de forma que os conceitos não-léxicos correspondem às tabelas e os conceitos léxicos aos campos das tabelas. Portanto, os conceitos `author` e `names` correspondem às tabelas `author` e `names` no BDR, assim como os respectivos elementos internos que equivalem às colunas da tabela. Maiores informações de como a ontologia e o esquema lógico relacional são gerados podem ser obtidas em (MELLO, 2007) e (SACCOL; PIVETA; ANDRADE, 2011).

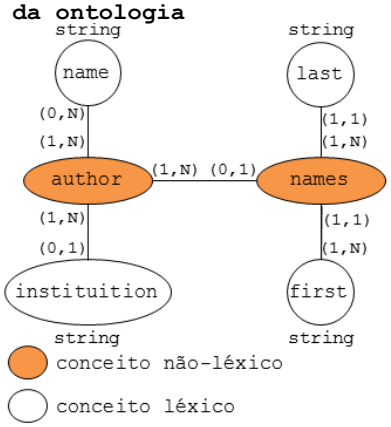
Doc_A.xml (fragmentado)

```
 ...
 5.<author>
 6. <names>
 7.  <last>Gerbarg</last>
 8.  <first>Darcy</first>
 9. </names>
10. <instituition>CITI</instituition>
11.</author>...
```

Doc_B.xml (fragmentado)

```
...
5.<writer>
6. <name>W. Stevens</name>
7.</writer>...
```

Representação gráfica da ontologia



Esquema lógico relacional

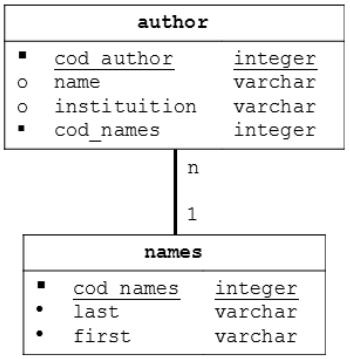


Figura 2.4. Equivalência entre documentos XML, ontologia e o esquema lógico relacional.

Para documentar a equivalência entre os documentos XML, a ontologia e o esquema lógico relacional, foi convencionado um documento de mapeamento como artefato de representação das equivalências, conforme ilustrado na Figura 2.5. O documento de mapeamento completo pode ser visto no Apêndice A.

```
...
2.<mapping>                                          conceito da ontologia
105. <concept name="author">
106.  <source id="Doc_A.xml">
107.   <xpath>/bibliography/book/author</xpath>      caminho XPath das fontes
108.  </source>                                      de dados XML
109.  <source id="Doc_B.xml">
110.   <xpath>/bibliography/book/writer</xpath>
111.  </source>
112.  <relational>                                   representação relacional
113.   <type>table</type>                            para conceito não-léxico
114.   <name>author</name>
115.  </relational>
116. </concept>...
160.  <relational>                                   representação relacional
161.   <type>column</type>                           para conceito léxico
162.   <name>name</name>
163.   <table>author</table>
164.   <domain>string</domain>
165.  </relational>
166. </concept>...
219.</mapping>
```

Figura 2.5. Documento de mapeamento parcial representando os documentos XML, a ontologia e o esquema lógico relacional referentes à Figura 2.4.

O documento de mapeamento baseia-se no elemento `<concept>`, linha 105, para representar os conceitos presentes na ontologia. Em cada conceito são referenciadas as equivalências em cada documento XML e no esquema relacional. Cada documento é identificado pelo atributo `id` do elemento `<source>`, ambos na linha 106. O conteúdo de cada fonte XML é apontado pelo caminho XPath na linha 107. O elemento `<relational>`, na linha 112, indica a equivalência da instância XML no esquema relacional de banco de dados, ou seja, se representa tabela ou coluna.

2.1.3. Componente XMap

O componente XMap (AVELAR; SACCOL, PIVETA, 2012) tem a funcionalidade de mapear e armazenar o conteúdo dos documentos XML no banco de dados. Para isso, recebe os documentos XML, o documento de mapeamento dos conceitos (gerado por CMap) e o respectivo esquema relacional (gerado por OntoRel) e gera o *script* de inserção do conteúdo XML em linguagem SQL DML para execução no BDR. O processo de inserção dos dados XML no banco de dados está estruturado em cinco níveis, conforme ilustrado na Figura 2.6.

O primeiro nível associa os dados de entrada, incluindo os documentos XML, a ontologia e o esquema relacional. A representação integrada das entradas é responsável por especificar as equivalências entre conceitos, associando e mapeando nos três modelos (XML, ontologia e relacional). Como saída, tem-se um documento de mapeamento, que é o mesmo gerado por CMap (NEGRINI et al., 2012).

No terceiro nível são interpretados os caminhos de armazenamento, presentes no documento de mapeamento, com o descritor XML, que é um arquivo auxiliar gerado pelo programador. O resultado é criado na forma de objetos em memória no nível 4.

Os arquivos XML são interpretados pelo gerenciador de acordo com os objetos de mapeamento. Caso ocorrer um conflito, um processo de resolução de conflitos é responsável por resolver a inconsistência dos dados. São tratados conflitos de: nomenclatura, de estrutura, de elemento inexistente e de representação. Um exemplo prático de identificação de conflitos é apresentado na seção 3.6.2. Maiores detalhes de como os conflitos são resolvidos podem ser encontrados em (AVELAR; SACCOL, PIVETA, 2012).

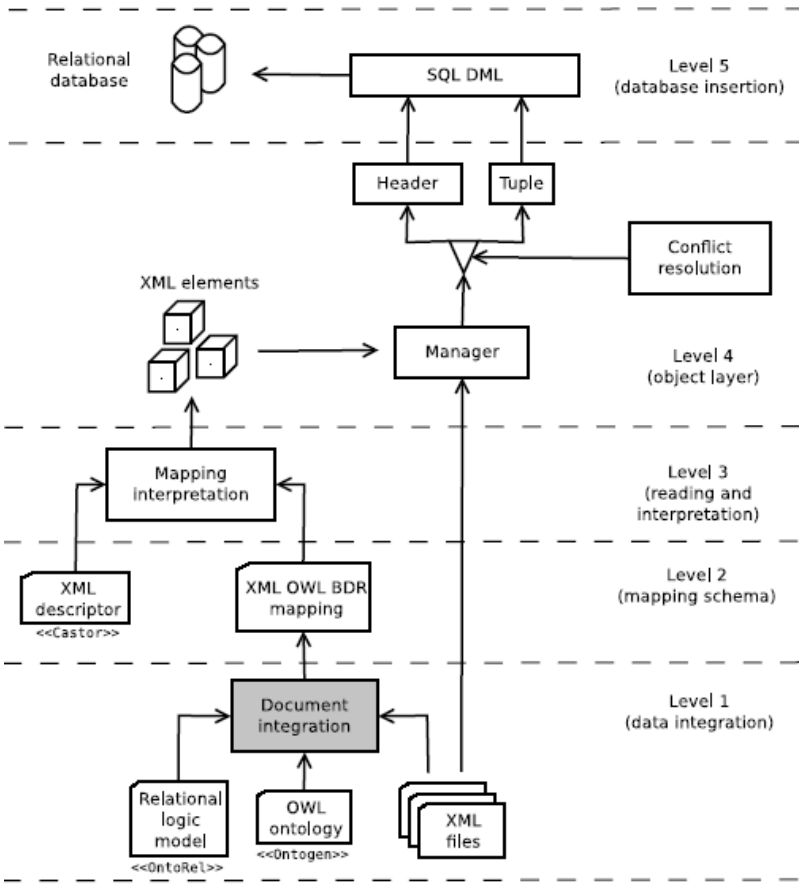


Figura 2.6. Arquitetura do componente XMap (AVELAR; SACCOL, PIVETA, 2012).

A partir do mapeamento dos objetos em memória e dos arquivos XML, o gerenciador tem a tarefa de gerar os objetos de cabeçalho para a inserção no banco de dados, juntamente com as respectivas tuplas. Finalmente, no quinto nível é gerado o *script* SQL DML para inserir o conteúdo dos documentos XML em tabelas e colunas no BDR.

Resumidamente, a funcionalidade do componente XMap no *framework* X2Rel é interpretar o documento de mapeamento, identificar o conteúdo nas fontes de dados XML através do caminho XPath e associar com o esquema lógico relacional através dos conceitos descritos pela ontologia. Durante o processo de mapeamento, conflitos de inserção podem ocorrer, os quais são tratados e resolvidos em tempo de execução.

Neste trabalho, o XMap foi incorporado para desempenhar a função de compor as operações de atualização do tipo "`insert`", assim, tornando-o operacional tanto no processo de inserção dos documentos originais no BDR quanto no processo de atualização. Na seção 3.7.3 é descrito o processo de montagem das instruções de inserção em SQL.

2.2. Manutenção de Visões Relacionais de Documentos XML

Neste trabalho considera-se o BDR como uma visão materializada dos documentos XML. Sempre que os arquivos XML são atualizados, as modificações devem ser propagadas para o banco de dados. Isto se refere a um problema bem conhecido na área de banco de dados, o problema de manutenção de visões.

Uma visão materializada pode ser entendida como o armazenamento físico de dados, obtidos a partir das fontes originais. Consultas podem ser submetidas às visões, sem o acesso direto às fontes de dados que a originaram. Já a manutenção da visão é o processo que garante que o estado da visão reflita o estado dos dados fonte. Na literatura é possível encontrar algumas definições para o problema de manutenção de visões.

Em (GUPTA; MUMICK, 1995), a manutenção (ou atualização) de visão é definida como o processo de atualizar uma visão em resposta a modificações nos dados fonte. Basicamente, este processo permite que você modifique uma parte da visão de acordo com às modificações nas fontes de dados.

De maneira semelhante, (CHEN; LIÃO, 2010) argumenta que a atualização de visões ainda é um problema em aberto, que significa traduzir as modificações nos dados de origem em uma sequência de atualizações sobre a visão. O problema é que nem sempre há uma correspondência direta entre o estado dos dados de origem e o estado da visão. Dessa forma, a solução para o problema de atualização ainda depende do contexto de aplicação.

Duas questões são consideradas importantes no processo de manutenção de visões relacionais que armazenam dados XML: (i) como identificar uma modificação nos dados XML e; (ii) como refletir as operações de atualização XML para a visão. A primeira questão remete ao problema de gerenciamento de modificações, que pode ser resolvido com o uso de técnicas para detecção de diferenças, que já são bastante utilizadas neste contexto. A segunda questão é considerada mais complexa, pois está diretamente relacionada à maneira como propagar modificações dos dados de origem para os dados da visão.

Uma alternativa possível é reprocessar todo o conteúdo da visão em resposta às modificações nos dados fontes; dessa forma, mantém-se a visão atualizada. No entanto, esta estratégia pode se tornar inviável, até mesmo não justificando a existência da visão.

De maneira geral, técnicas para manutenção de visões que envolvem dois modelos de dados procuram definir alguma estratégia para referenciar os dados entre um modelo e outro

(neste caso, XML e relacional). A definição de identificadores pode ser uma alternativa eficaz, sendo que os mesmos permitem referenciar os dados fonte com os dados armazenados na visão. No entanto, este processo acrescenta maior complexidade ao projeto, ainda mais tratando de representações XML heterogêneas, onde é necessário um processo de verificação de equivalências entre instâncias dos dois modelos.

Neste trabalho, o problema de propagação de modificações nos dados fonte para a visão é solucionado através do rastreamento com chaves. Isto levanta uma questão importante em projetos de banco de dados, que é a definição e o uso das restrições de chave, que também desempenha um papel fundamental na atualização de visões relacionais de dados XML.

2.3. Algoritmos para Detecção de Diferenças em Documentos XML

Devido a aceitação da linguagem XML como um padrão para a representação de dados semiestruturados, uma grande quantidade de documentos XML tem sido gerados diariamente pelos mais diversos tipos de aplicações. Grande parte destes documentos tem a finalidade de armazenar dados; entretanto, como em qualquer base de dados, os documentos XML podem sofrer modificações ao longo do tempo, sendo necessário detectar e gerenciar essas modificações. Além do mais, o interesse pode não estar em uma versão específica do documento, mas nas modificações que esse sofreu ao longo do tempo.

Diante disso, esforços têm sido feito no sentido de detectar modificações entre documentos XML. Exemplo disso são os algoritmos de *Diff*, como o XyDiff (COBÉNA; ABITEBOUL; MARIAN, 2002), desenvolvido especificamente pra detectar modificações em dados XML. O XyDiff foi inicialmente desenvolvido para um contexto de detecção de modificações em grandes volumes de dados, mais especificamente para um projeto que envolve *Data Warehouse*. Por este motivo, o algoritmo concentra-se em uma estratégia que busca melhorar a gestão do tempo e espaço em memória.

De maneira geral, os algoritmos de *Diff* compartilham um propósito comum, que é o de obter uma sequência finita de operações de edição para converter um documento em outro, também chamado de delta. Para representar o delta, cada algoritmo segue uma estrutura particular, que pode ser voltada para uma análise direta pelo usuário ou para a análise por aplicações. O formato de representação do delta é um aspecto fundamental, onde se ganha ou

perde-se importância de acordo com o objetivo de aplicação. Por exemplo, se a aplicação envolver o uso de banco de dados, pode ser interessante um delta que especifique um modelo de operações simplificado, que não inclua operações de realocação de nodos na árvore XML. Esse modelo de operações, geralmente, é suportado por algoritmos que não consideram a ordenação da árvore XML.

Uma proposta para detecção de diferenças entre árvores não-ordenadas, a qual considera apenas a relação ancestral-descendente, é apresentada em (WANG; DEWITT; CAI, 2003), através do algoritmo X-Diff. Segundo os autores, a detecção de diferenças em árvores não-ordenadas é substancialmente mais difícil que no modelo ordenado; porém, o resultado obtido é mais preciso. Para gerar o delta, o X-Diff gera uma assinatura única para cada nodo, buscando a correspondência de custo mínimo entre os nodos de uma árvore e outra. Aquele nodo que não for correspondente representa uma modificação no documento XML e receberá uma operação de edição.

Já para detectar modificações em árvores ordenadas, onde é considerada tanto a relação ancestral-descendente quanto a ordem entre os nodos irmãos, foram desenvolvidos algoritmos como o XyDiff, discutido anteriormente, e o DiffX (AL-EKRAM; ADMA; BAYSAL, 2005). O DiffX busca a correspondência entre os nodos a partir do mapeamento de fragmentos isolados na árvore, buscando identificar o casamento entre os fragmentos mais amplos. Este processo é realizado recursivamente, da raiz da árvore para os nodos filhos, até que todos os nodos tenham sido casados. Seu modelo de operações especifica somente as operações de deleção, inserção e movimentação. Para gerar o delta, a estratégia usada é de deletar todos os nodos não correspondentes, inserir todos os nodos correspondentes e mover os nodos correspondentes que possuem ancestrais ou posição não correspondente.

Também existem ferramentas que detectam modificações independentemente do modelo de ordenação da árvore, se ordenada ou não-ordenada. Exemplos dessas aplicações são o DeltaXML (FONTAINE, 2001) e o XML Diff and Tools (MICROSOFT, 2004). O DeltaXML incorpora um conjunto de ferramentas comerciais, desenvolvidas pela Monsell EDM Ltd (MONSELL, 2014), que permitem comparar, mesclar e sincronizar alterações em documentos XML. Uma característica interessante do DeltaXML é o suporte à comparação entre duas e três versões de documentos, o que permite detectar o delta entre duas versões e comparar esse delta com um documento. Já o conjunto de ferramentas XML Diff and Tools oferece suporte às operações básicas de edição mais a operação de movimentação. No entanto, não é uma ferramenta distribuída livremente, portanto, existe uma carência em

relação à documentação. Além disso, ela não consegue comparar arquivos que diferem estruturalmente.

Nota-se que cada algoritmo foca em determinadas características para detectar diferenças. Por terem objetivos semelhantes, alguns algoritmos podem apresentar menores variações de características em relação a outros. Desse modo, definir o algoritmo adequado ao contexto de aplicação pode representar um desafio. Neste trabalho de dissertação, o algoritmo de *Diff* é selecionado com base no contexto de banco de dados. Logo, o X-Diff mostrou-se mais adequado pelo aspecto do seu modelo de árvore e operações, conforme detalhado a seguir.

2.3.1. Algoritmo X-Diff

Duas características tornam o algoritmo X-Diff (WANG; DEWITT; CAI, 2003) adequado à proposta deste trabalho: (i) o modelo não-ordenado de árvore XML e (ii) o modelo de operações simplificado. A primeira característica é determinada pelo ambiente de armazenamento de dados XML em BDR, onde as tuplas nas tabelas do banco de dados não são ordenadas; portanto a relação de ordem dos nodos no documento XML não é relevante. A segunda característica mostra-se adequada ao contexto de tradução das operações de edição XML para operações de atualização sobre o BDR, pois o X-Diff especifica somente as operações básicas de atualização (inserção, deleção e atualização), que são as mesmas especificadas pela linguagem SQL.

Para gerar o delta, o X-Diff segue algumas etapas, onde, inicialmente, o algoritmo realiza a leitura de dois documentos XML de entrada e os transforma em duas estruturas de árvores DOM (*Document Object Model*) (HEGARET; WHITMER; WOOD, 2006). Essa estrutura utiliza três tipos de nodos:

- *Elemento* – nodo não-folha com nome;
- *Atributo* – nodo folha com nome e valor;
- *Texto* – nodo folha com valor.

Durante o processo de leitura, o X-Diff gera uma assinatura para cada nodo na estrutura da árvore. A assinatura consiste num identificador único para o nodo, sendo formada pelo nome do nodo juntamente com o nome dos ancestrais deste nodo. Essa assinatura é usada

para encontrar a correspondência entre os nodos de uma árvore e outra. Com esta estratégia, evita-se que nodos de tipo diferentes sejam comparados, o que causaria uma computação desnecessária.

Na Tabela 2.1 são exemplificadas as assinaturas do elemento `title` (linha 4), do atributo `@year` (linha 3) e do texto do elemento `title` (linha 4), todos referentes ao documento `New.xml` da Figura 1.2 .

Tabela 2.1. Exemplos de assinatura de nodos no X-Diff.

Nodo	**Assinatura do nodo**
linha 4 (`title`)	`/book/title/$ELEMENT$`
linha 3 (`@year`)	`/book/@year/$ATTRIBUTE$`
linha 4 (`"The Economics of Technology..."`)	`/book/title/$TEXT$`

No processo de correspondência dos nodos, o X-Diff usa uma estratégia para encontrar uma correspondência de custo mínimo entre duas árvores. Para isso, dada duas árvores XML, o algoritmo faz uma busca recursiva, desde a raiz do documento até os nodos folha, para encontrar nodos cuja assinatura seja equivalente. Dessa forma, a correspondência entre os nodos ocorre somente se a assinatura dos nodos for idêntica, se os nodos ancestrais já forem correspondentes e se existir correspondência entre os nodos raiz.

Na etapa de geração do delta, o X-Diff gera um *script* com operações de edição de custo mínimo, com base na correspondência entre os nodos na etapa anterior. Uma análise aprofundada sobre delta com custo mínimo pode ser obtida em (WANG; DEWITT; CAI, 2003). Especificamente, o X-Diff define três operações básicas de edição que podem operar sobre subárvores e nodos folha para inclusão e exclusão. Operações de modificação somente são aceitas em nodos do tipo texto. Caso um elemento ou atributo tenha seu nome modificado, o X-Diff detecta como uma exclusão seguida de uma inclusão, mesmo que a subárvore mantenha-se inalterada. Conteúdos de texto e valores de atributos podem ser modificados.

As operações reconhecidas pelo X-Diff são expressas como segue (WANG; DEWITT; CAI, 2003):

- *`Insert(x(name, value), y)`* – insere um nodo folha *`x`* com o nome *`name`* e valor *`value`*. O nodo *`x`* é inserido como filho do nodo *`y`*.
- *`Delete(x)`* – deleta o nodo folha *`x`*.

- *Update(x, new-value)* – atualiza o nodo *x* como o novo valor *new-value*.
- *Insert(T_x, y)* – insere uma subárvore T_x (tendo o nodo *x* como raiz) como filha do nodo *y*.
- *Delete(T_x)* – deleta a subárvore T_x (tendo o nodo *x* como raiz).

Conforme as etapas descritas anteriormente, pode-se resumir que o X-Diff gera o delta de acordo com as seguintes observações: a) caso existir correspondência entre os nodos, então nenhuma operação é adicionada; b) caso não existir correspondência entre nodos do tipo elemento ou atributo, então o delta recebe operações de inserção ou deleção e; c) caso não existir correspondência apenas em nodos do tipo texto (nodos folha), então o delta recebe a operação de atualização (*update*).

Apesar do X-Diff ter uma boa performance na detecção de modificações entre documentos pequenos, em se tratando de documentos em grande escala o algoritmo tem uma queda de performance, conforme estudos apresentados em (LEONARDI; BHOWMICK, 2005). Esta limitação se explica pela estratégia que o algoritmo usa para representar os documentos em árvores DOM (*Document Object Model*), sendo que estas armazenam as informações em memória, logo, estão limitadas ao tamanho de memória da máquina.

2.4. Esquemas de Rotulagem em XML

Alguns problemas relacionados à manipulação de dados XML, tais como a recuperação e atualização dos dados, ainda são questões em aberto. Esforços têm sido feitos no sentido de armazenar dados XML em BDR (TATARINOV et al., 2002) (FLORESCU; KOSSMANN, 1999). Assim, os dados antes representados em uma estrutura XML podem ser manipulados através de um SGBD relacional. Nesse sentido, os esquemas de rotulagem XML têm uma função importante, que é a de atribuir identificadores aos nodos no documento XML. Com isso, pode-se reconstruir os documentos em seu formato original, seguindo o relacionamento entre os identificadores.

Os esquemas de rotulagem podem ser entendidos como identificadores que permitem preservar a ordem e a relação entre os nodos na estrutura do documento XML. Weigel, Schulz e Meuss (2005) argumentam que um esquema de rotulagem é um resumo estrutural descentralizado de um conjunto específico de relacionamentos. Tais relacionamentos são

classificados como pai-filho, ancestral-descendente e entre irmãos (que também permite determinar a ordem do nodo). Para permitir o relacionamento, cada nodo do documento recebe um identificador único, de modo que qualquer relação entre os nodos pode ser inferida a partir do identificador.

Conceitualmente, os esquemas de rotulagem são classificados em duas categorias: baseados em intervalo (DIETZ, 1982) (ZHANG et al., 2001) (LI; MOON, 2001) (AMAGASA, et al., 2003) e baseados em prefixo (TATARINOV et al, 2002) (O'NEIL et al., 2004) (XU et al., 2009). A primeira representa a forma mais simples, onde o rótulo codifica a posição e a extensão de uma subárvore através de uma sequência de deslocamento de nodos. Já na segunda, também conhecida como rotulagem baseada em caminho, o rótulo codifica o caminho da raiz do documento até o nodo com uma sequência para denotar exclusivamente o ancestral do nodo nesse caminho.

Tanto o esquema baseado em intervalo (*offset*) quanto o baseado em prefixo são amplamente utilizados pelos mais variados tipos de aplicações, sendo que o esquema adequado deve ser selecionado de acordo com a finalidade de uso. Por exemplo, em uma aplicação que exige o identificador do nodo como constante, ou seja, que o rótulo não mude de valor, é conveniente utilizar um esquema que não tenha que reconstruir o rótulo a cada modificação no documento.

A rotulagem baseada em intervalo foi o primeiro esquema proposto; seu objetivo inicial era determinar as relações ancestrais e manter uma árvore estruturada (DIETZ, 1982). Com o passar do tempo, este contexto inspirou métodos para rotulagem de dados semiestruturados. Para exemplificar, considera-se a abordagem apresentada em (ZHANG et al., 2001), que considera a posição inicial e final do nodo para atribuir o rótulo. Nesta abordagem, cada nodo do tipo elemento recebe um rótulo com posição inicial e final, sendo que o início e o fim definem o intervalo que contém todos os descendentes. No esquema baseado em intervalo, o nodo XML corresponde a uma *substring* de uma *string* do documento. As *substrings* podem ser naturalmente identificadas pelas coordenadas de região, que são interpretadas com um par de números inteiros (posição inicial, posição final) contando a partir da raiz do documento.

A concepção de esquemas de rotulagem que cumprem todos os critérios acaba sendo um problema desafiador. A maioria dos trabalhos disponíveis na literatura ainda não satisfaz o critério de atualização e requerem a reconstrução do rótulo ao atualizar os documentos XML.

Algumas soluções mais dinâmicas (O'NEIL et al., 2004) (LI; LING; HU, 2008) têm sido propostas; no entanto, o problema de atualização ainda é resolvido parcialmente.

Em (TATARINOV et al., 2002), são apresentados três importantes esquemas de rotulagem de nodos baseado em prefixo, que são usados como base em muitas abordagens: Ordem Global, Ordem Local e Ordem de Dewey:

- Ordem Global. A árvore XML é percorrida em pré-ordem e um número identificador é atribuído a cada nodo, de maneira que represente sua posição absoluta no documento. No entanto, no caso de inserção de um novo nodo, todos os nodos posicionados após o nodo inserido precisam ter seus rótulos reconstruídos. Além disso, torna-se impossível inferir o relacionamento pai-filho e ancestral-descendente.
- Ordem Local. Cada nodo recebe um rótulo identificador que representa sua posição relativa entre os nodos irmãos. O rótulo é composto pela posição do nodo combinado com seus ancestrais, de forma que um vetor de caminhos identifica a posição absoluta do nodo na estrutura do documento. Mesmo assim, ainda é difícil obter as relações pai-filho e ancestral-descendente.
- Ordem de Dewey. Cada nodo recebe um rótulo que representa o caminho da raiz do documento até o nodo, sendo que cada combinação do rótulo representa a ordem local de um nodo ancestral. Neste esquema, a relação ancestral-descendente é facilmente obtida; porém, em caso de inserção de um novo nodo, a reconstrução do rótulo é necessária.

Em (O'NEIL et al. 2004), a ordem de Dewey é usada para preservar as relações pai-filho e ancestral-descendente dos nodos. O esquema proposto mantém espaços entre os rótulos, de maneira que novos rótulos possam ser inseridos. No entanto, após os espaços reservados terem sido ocupados, muitos nodos terão que ser re-rotulados. Já em (SOLTAN, RAHGOZAR, 2006) é proposta uma abordagem semelhante à ordem de Dewey, com a diferença que o rótulo é atribuído a um agrupamento de nodos, ao invés de um rótulo individual a cada nodo. Com esta abordagem, mantem-se as relações pai-filho e ancestral-descendente. A desvantagem é que quando é inserido um novo nodo, todos os rótulos de nodos à direita, bem como seus descendentes, precisam ser re-rotulados.

Já em (DWEIB; AWADI; LU, 2009) é proposta a abordagem MDL (*Multi Dimension Link*), baseada em ordem Global para rotular nodos do tipo elemento ou atributo. Além do identificador de ordem global, cada nodo recebe outros três identificadores que fazem referência ao identificador do pai e do irmão à esquerda e à direita, formando um rótulo com

quatro dimensões. Esta estratégia possibilita que as relações pai-filho, ancestral-descendente e de irmãos possam ser inferidas a partir do rótulo. Ao contrário das abordagens anteriores, em caso de inserção ou realocação de um determinado nodo, o identificador global mantém-se constante (ou seja, não precisa ser re-rotulado), apenas o identificador dos nodos irmãos é atualizado.

Neste trabalho o interesse está no identificador global e na relação e ordem dos nodos que não deve ser alterada em consequência à uma atualização no documento. Nesse sentido, a abordagem MDL (DWEIB; AWADI; LU, 2009) e (DWEIB, 2013) mostrou-se a mais adequada ao contexto, sendo detalhada na seção 2.4.1.

2.4.1. Método MDL

Considerando que o valor das chaves para rastrear os dados XML no BDR é gerado a partir do identificador global (valor do identificador) do nodo, e que a reconstrução do documento no formato original é feita através da relação e ordem dos identificadores, a abordagem MDL (DWEIB; AWADI; LU, 2009) (DWEIB, 2013) mostrou-se adequada a este contexto. Tal abordagem baseia-se em um modelo de rotulagem de ordem global, onde as relações pai-filho, ancestral-descendente e de irmãos são preservadas, mesmo após uma atualização na estrutura do documento XML.

A abordagem MDL é uma modificação do esquema de Ordem Global apresentado em (TATARINOV et al., 2002), conforme descrito brevemente na seção 2.4, onde cada nodo no documento XML recebe um número identificador global que representa sua posição absoluta no documento. O diferencial deste modelo – em relação aos demais – é a utilização de informações da estrutura do documento para orientar o processo de rotulagem dos nodos, consequentemente a disponibilidade das informações de DTD (*Document Type Definition*) ou esquema XML (XML *Schema*) não são necessárias.

O modelo MDL foi proposto com base em requisitos para ambientes de armazenamento de documentos XML em BDR. Dessa forma, segue a estratégia de mapear e reconstruir o documento XML original, armazenado no BDR, a partir das informações do rótulo identificador de cada nodo. O uso de rótulos com multi-ligações dos nodos garante que

mesmo após inserções ou deleções, em qualquer posição do documento, a identidade do nodo mantém-se constante.

Para rotular os nodos, a árvore XML é percorrida em pré-ordem e um rótulo único é atribuído à cada elemento ou atributo, também chamado de *token*[3]. O diferencial deste modelo em relação aos apresentados em (TATARINOV et al., 2002), (SOLTAN; RAHGOZAR, 2006) e (TORSTEN, KEULEN; JENS, 2004) é que no caso de inserção de um novo *token*, o número do identificador é atribuído seguido do valor do último *token*; no caso de deleção, o identificador não é decrementado. Dessa forma, não há necessidade de re-rotular os nodos.

Cada nodo recebe um rótulo com quatro dimensões que identificam sua ordem e relação no documento. As dimensões são formadas pelo nodo de origem (*tokenID*), nodo irmão à esquerda (*leftID*), nodo pai (*parentID*) e nodo irmão à direita (*rightID*), conforme ilustrado na Figura 2.7.

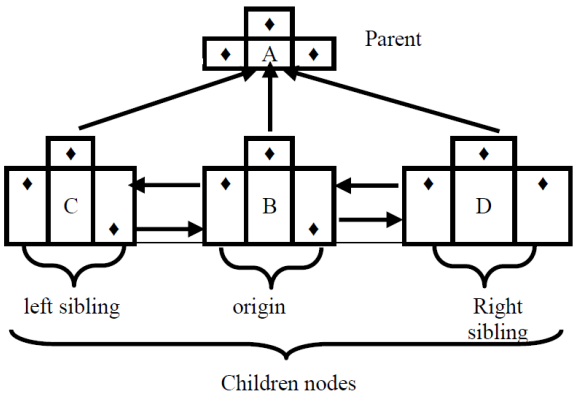


Figura 2.7.Visão do método MDL (DWEIB, 2013).

O *tokenID* e *parentID* são usados para manter a relação pai-filho e ancestral-descendente, enquanto o *leftID*, *rightID* e *tokenID* são usados para manter a ordem dos elementos e atributos relacionados como irmãos na estrutura do documento.

No caso de inserção de um novo *token* no documento, o modelo prevê a atribuição de um identificador seguido do valor máximo (último *tokenID*) do identificador global, conforme a regra:

`TokenID(T)` ← `maxTokenID + 1`

Onde:

O novo token `T` *recebe um identificador seguido do último token (*`maxTokenID`*)* .

[3] Nodo elemento ou atributo do documento XML.

O modelo também prevê regras para manutenção do rótulo do *token* para os casos de inserção ou deleção em alguma posição da árvore. Cada regra é destinada à manutenção de uma posição específica, como descrita:

a) Inserção de um novo *token* `T` à esquerda de uma subárvore, esquerda para `S1`:

```
1) RightID(T) ← tokenID(S1)
2) LeftID(T) ← Null
3) LeftID(S1) ← tokenID(T)
4) ParentID(T) ← ParentID(S1)
```

b) Inserção de um novo *token* `T` à direita de uma subárvore, direita para `S1`:

```
1) LeftID(T) ← TokenID(S1)
2) RightID(T) ← Null
3) RightID(S1) ← tokenID(T)
4) ParentID(T) ← ParentID(S1)
```

c) Inserção de um novo *token* `T` como filho de `S1`:

```
1) LeftID(T) ← Null
2) RightID(T) ← Null
3) ParentID(T) ← TokenID(S1)
```

d) Inserção de um *token* `T` entre dois irmãos, `S1` e `S2`[4]:

```
1) rightID(T) ← rightID(S1)
2) leftID(T) ← leftID(S2)
3) rightID(S1) ← tokenID(T)
4) leftID(S2) ← tokenID(T)
5) ParentID(T) ← ParentID(S2)
```

e) Inserção de um novo *token* `T` como pai de `S1`:

```
1) LeftID(T) ← LeftID(S1)
2) RightID(T) ← RightID(S1)
3) ParentID(T) ← ParentID(S1)
4) LeftID(S1) ← Null
5) RightID(S1) ← Null
6) ParentID(S1) ←  TokenID(T)
7) TreeLevel(T) ← TreeLevel(S1)
8) Buscar todos os descendentes e incrementar 1 nível
```

Em caso de deleção de um *token* existente, o valor dos *`TokenID`* não serão alterados (isto é, não são decrementados). O processo de manutenção segue as seguintes regras:

a) Deleção de um *token* `T` entre dois irmãos, `S1` e `S2`:

```
1) RightID(S1) ← RightID(T)
```

[4] **S1** e **S2** correspondem ao *token* candidato à esquerda ou à direita.

2) *LeftID(S2)* ← *LeftID(T)*

b) Deleção de um *token* `T` à esquerda de uma subárvore, esquerda de `S1`:

1) *LeftID(S1)* ← *Null*

c) Deleção de um *token* `T` à direita de uma subárvore, direita de `S1`:

1) *RightID(S1)* ← *Null*

d) Deleção de um elemento complexo;

A deleção de uma subárvore pode ser tratada como um único *token*, por um dos três casos anteriores, mas todos seus descendentes também serão deletados.

2.5. Trabalhos Relacionados

Partindo do contexto em que os dados XML estão armazenados no BDR, sendo que o banco de dados representa uma visão materializada dos dados XML, a atualização do banco de dados nos remete a um problema bastante conhecido na área de banco de dados que é a atualização de visões. Em um cenário ideal, as modificações nos dados XML poderiam ser propagadas através de uma condição que usa chaves. Desse modo, os dados armazenados na visão podem ser facilmente rastreados. No entanto, essa situação não é o que normalmente ocorre; é comum encontrar documentos XML com uma estrutura própria de chaves ou sem nenhuma estrutura. Sendo assim, a geração de chaves XML direciona a um contexto de aplicação que pode ser uma solução satisfatória.

Para resolver o problema de manutenção da visão relacional dos dados XML, este trabalho apresenta uma abordagem baseada na detecção de diferenças e rastreamento de dados através de chaves, que pode ser utilizada para o aperfeiçoamento de técnicas tradicionais de armazenamento e atualização de visões relacionais de dados XML. A finalidade do uso de chaves é permitir que os dados da visão possam ser referenciados aos dados fonte, independente da maneira de como os arquivos XML estão estruturados. Assim, essa abordagem atribui um identificador comum entre as entidades dos modelos de dados (XML e relacional), de maneira que uma modificação nos dados fonte possa ser rastreada na visão, usando a chave como condição.

O problema de propagação de modificações para visões não é uma questão nova, mas com a disseminação da linguagem XML para representação de dados, este problema passou a ser explorado no contexto de atualizar visões relacionais de dados XML e visões XML de BDR. Embora existam algumas técnicas para propagar modificações de XML para dados relacionais, apenas algumas delas são baseadas no rastreamento dos dados através de chaves. No contexto deste trabalho, a diversidade de estrutura dos documentos XML não permite usar chaves XML que se baseiam na estrutura individual dos documentos.

De forma a proporcionar uma visão geral dos trabalhos relacionados, a seção 2.5.1 apresenta as principais propostas que tratam o problema de manutenção de visões. A seção 2.5.2 faz um comparativo entre os trabalhos apresentados e a proposta deste trabalho.

2.5.1. Abordagens para Manutenção de Visões

Em (WANG; RUNDESTEINER; MANI, 2006) é apresentada uma abordagem que considera as chaves estrangeiras das relações para determinar se a tradução da atualização está correta. A metodologia empregada classifica uma modificação sobre a visão como intraduzível, condicional ou incondicionalmente traduzível. Para classificar uma dada atualização em alguma das três categorias, os autores apresentam um algoritmo baseado em grafos. Este algoritmo inclui duas etapas principais: a primeira atribui um rótulo para cada nodo do documento XML, indicando suas propriedades de atualização; a segunda usa os rótulos para classificar uma modificação sobre a visão em uma das três categorias (intraduzível, condicional ou incondicionalmente traduzível).

(BRAGANHOLO; DAVIDSON; HEUSER, 2006) propõe uma abordagem que propaga modificações em visões XML, definidas a partir de um BDR, para um conjunto de operações de atualização sobre o banco de dados. Para atualizar o BDR, a visão XML é mapeada para uma visão relacional correspondente. A partir daí, técnicas de atualização de visões relacionais já existentes podem ser empregadas para atualizar o banco de dados. Nesta abordagem, é adotada a técnica apresentada em (DAYAL; BERNSTEIN, 1982) que traduz as modificações sobre a visão para o banco de dados.

Em (VARGAS, 2006) é apresentada uma extensão para o trabalho de (BRAGANHOLO; DAVIDSON; HEUSER, 2006), sendo que o diferencial está no processo

de detecção das modificações sobre os dados XML. Para detectar as modificações, são usadas técnicas de detecção de diferenças em documentos XML, onde o documento XML não-modificado é comparado com o modificado, gerando o delta entre as duas versões. A partir do delta, as operações de modificação são propagadas para o BDR utilizando uma linguagem de atualização XML apresentada em (BRAGANHOLO; DAVIDSON; HEUSER, 2006).

(CHOI et al. 2008) investiga o problema de atualização de visões XML definidas recursivamente a partir de dados relacionais. Os autores focam em uma solução baseada na análise de nível do esquema para determinar se uma atualização sobre a visão é traduzível e encontrar a tradução (se houver) considerando as restrições do esquema XML. Portanto, dada a visão XML de um BDR, as modificações precisam ser propagadas para as tabelas originais sem comprometer a integridade dos dados XML nem dos dados relacionais. O objetivo principal é investigar os efeitos colaterais que são traduzidos através de uma atualização sobre a visão. A diferença deste trabalho em relação aos anteriores é que as atualizações com efeito colateral não são descartadas, mas consultadas junto ao usuário final para obter um melhor resultado.

A relação entre o destino da operação de modificação e os dados de origem motivou (FEGARAS, 2010) a propor uma abordagem baseada no rastreamento de procedência de dados. Através desta abordagem é possível determinar se uma operação de modificação sobre a visão, expressa através de uma linguagem de atualização XML, pode ser rastreada de volta à sua origem e caso for, são geradas as operações de atualização correspondentes, permitindo que as tabelas no BDR sejam atualizadas e o resultado obtido seja o mesmo de uma modificação direta no documento XML.

Dentre as abordagens que tratam do mapeamento e armazenamento de documentos XML em BDR, destaca-se a abordagem de (DWEIB; AWADI; LU, 2009). Tal abordagem usa duas tabelas relacionais para armazenar os dados dos documentos XML, sendo uma tabela chamada *Documents* destinada ao armazenamento da estrutura dos documentos, e uma tabela *Tokens* para armazenamento das informações dos nodos e seus identificadores, usados para reconstruir o documento de origem. Em caso de modificação no documento XML, a abordagem usa como base as multi-dimensões que ligam o nodo para garantir que o identificador do nodo mantenha-se com o valor constante. A abordagem apresenta diversas regras para manutenção dos identificadores, as quais foram de interesse no decorrer desta dissertação.

2.5.2. Comparativo dos Trabalhos Relacionados

Diversas abordagens foram desenvolvidas para propagar modificações para visões. Conforme apresentado na seção 2.5.1, pesquisas propõem soluções para o problema de atualização de visões.

Dentre as abordagens discutidas, a manutenção de visões relacionais tem sido bastante utilizada para tratar o problema de atualização de visões XML. Dessa forma, técnicas antigas de atualização de visões relacionais são usadas para solucionar o novo problema de atualização de visões XML, como em (WANG; RUNDESTEINER; MANI, 2006) e (BRAGANHOLO; DAVIDSON; HEUSER, 2006). Nesse caso, a visão XML é definida sobre o BDR e posteriormente espelhada para uma ou mais visões relacionais, possibilitando que a propagação das modificações para o banco de dados seja feita sobre o mesmo modelo de dados (ou seja, da visão relacional para o BDR), delegando o problema de atualização ao SGBD. Por outro lado, o trabalho proposto trata da atualização de visões relacionais definidas a partir de documentos XML estruturalmente distintos, o que acrescenta maior complexidade ao processo de identificação e mapeamento dos dados alvo (que devem ser atualizados) para as correspondentes tabelas relacionais.

Os trabalhos relacionados apresentam soluções para a criação das visões, seja diretamente a partir dos documentos XML (DWEIB; AWADI; LU, 2009), seja a partir do BDR (WANG; RUNDESTEINER; MANI, 2006) (BRAGANHOLO; DAVIDSON; HEUSER, 2006) (CHOI et al. 2008). No caso de propostas que partem estritamente do BDR, as chaves para rastreamento dos dados são compartilhadas com as tabelas relacionais, sem a necessidade de uma estratégia para gerar as chaves de maneira artificial. A estratégia de geração de chaves apresentada neste trabalho baseia-se no conceito do nodo XML no esquema relacional. Dependendo do conceito, o elemento XML receberá o atributo chave para rastrear os dados XML com as tuplas no BDR. No entanto, a proposta limita-se ao rastreamento da chave primária, não tratando a geração e propagação das chaves estrangeiras para as relações.

De forma semelhante, o trabalho de (VARGAS, 2006) fez a detecção de modificações nos documentos XML através de algoritmos de *Diff* já existentes. Nesse aspecto, foi possível perceber que o processo de identificação de modificações através de algoritmos de *Diff* é uma prática comum em ambientes cujas alterações são realizadas pelo usuário diretamente na fonte

do documento XML, sem o uso de uma linguagem de atualização XML específica. A vantagem da abordagem proposta neste trabalho é que as operações presentes no delta são traduzidas para uma sequência de operações de atualização (*insert*, *delete* ou *update*) em linguagem SQL (considerada padrão para BDR), diferentemente do trabalho analisado que depende de uma linguagem de atualização própria, tornando a abordagem dependente de uma tecnologia específica.

De outra maneira, (FEGARAS, 2010) usou o rastreamento de dados entre o modelo XML e relacional para determinar a procedência de uma operação de modificação com os dados de destino. Nesses casos, foi abordada a tradução de uma operação de atualização expressa através de uma linguagem XML para a linguagem SQL. No entanto, mesmo a abordagem proposta não prevendo o rastreamento de uma operação de atualização de volta aos dados de origem, é possível chegar aos dados fontes através do rastreamento da chave primária das tabelas com a chave XML previamente gerada.

2.6. Considerações Finais

O *framework* X2Rel é o ambiente usado para contextualizar a propagação de modificações para a visão através do rastreamento de dados por chaves. Diante da arquitetura do X2Rel, a abordagem deste trabalho foi definida pelo componente DBUpdater. Especificamente, a tarefa do DBUpdater é identificar e propagar modificações nos documentos XML para o BDR. Para isso, o DBUpdater utiliza o documento de mapeamento gerado pelo componente CMap para identificar a equivalência das instâncias XML no esquema lógico relacional gerado por OntoRel. Espera-se, como artefato de saída do componente DBUpdater, um *script* SQL com as instruções de atualização (*insert*, *delete* e *update*) para ser executado sobre o banco de dados.

De maneira a identificar as modificações realizadas nas fontes de dados XML, a detecção de diferenças através de algoritmo de *Diff* foi a alternativa mais compatível para o contexto de manutenção da visão. Para detectar o delta, duas versões são mantidas para cada documento fonte: a primeira corresponde à versão original e a segunda à versão modificada. Dessa forma, o problema de detecção das modificações resume-se em comparar duas versões de um mesmo documento XML. O X-Diff mostrou-se o algoritmo mais adequado ao contexto

do trabalho, pois, além de não considerar a ordenação da árvore XML, também possui um modelo de operações simplificado, semelhante ao modelo de operações de atualização da linguagem SQL.

De maneira a permitir o rastreamento dos dados XML armazenados no banco de dados relacional, a geração e atribuição de rótulos identificadores aos nodos no documento mostrou-se uma alternativa viável para a geração das chaves. Foram apresentadas diversas abordagens para gerar os rótulos; entretanto, considerando que o valor das chaves é gerado a partir do identificador global do nodo, é necessário que o identificador mantenha-se constante mesmo após alterações na estrutura do documento XML. Neste contexto, o esquema MDL mostrou-se mais adequado, pois permite manter as relações pai-filho, ancestral-descendente e de irmãos. Além disso, o documento original pode ser reconstruído a partir da relação e ordem dos identificadores.

3. PROPOSTA DE PROPAGAÇÃO DE MODIFICAÇÕES

Os capítulos anteriores abordaram o problema de manutenção de visão relacional de documentos XML, bem como o referencial teórico necessário para o entendimento da proposta de solução. Foram apresentados os conceitos e tecnologias relevantes para o entendimento do processo de manutenção de visões, detecção de diferenças e esquemas de rotulagem de dados XML. Partindo da arquitetura do *framework* X2Rel, a proposta deste trabalho está na abordagem do componente DBUpdater para detectar e propagar modificações no conteúdo dos documentos XML para o BDR.

3.1. Visão Geral

O foco deste trabalho é propor uma abordagem para identificar e propagar modificações de documentos XML armazenados em uma visão relacional materializada. Dessa forma, se o documento XML sofrer modificações de inclusão, alteração ou exclusão de conteúdo, a devida propagação das alterações para as tabelas relacionais é realizada de forma transparente para o usuário. Para isso, são consideradas as informações do esquema relacional e do mapeamento dos conceitos, originados a partir dos demais componentes do *framework* X2Rel.

Para exemplificar o processo de detecção e propagação de modificações, na Figura 3.1 há duas versões do documento `Doc_A.xml`, sendo, `Old.xml` a original e `New.xml` a modificada.

Documento Old.xml (fragmentado)

```
...
<book year="2005">
 <title>Advanced Program...</title>
 <author>
  <names>
   <last>Stevens</last>
   <first>W.</first>
  </names>
 </author>
 <publisher>Addison-Wesley</publisher>
 <edition>2</edition>
 <price>42.25</price>
</book>...
```

Documento New.xml (fragmentado)

```
...
<book year="2005">
 <title>Advanced Program...</title>
 <author>
  <names>
   <last>Stevens</last>
   <first>W.</first>
  </names>
  <names>
   <last>Rago</last>
   <first>Stephen</first>
  </names>
 </author>
 <publisher>Addison-Wesley</publisher>
 <edition>2</edition>
 <price>53.58</price>
</book>...
```

Figura 3.1. Versão original e modificada do documento `Doc_A.xml`.

Comparando a versão original e modificada da Figura 3.1, é possível identificar a inclusão de um novo autor (linha `23` de `New.xml`) do livro "`Advanced Program...`" e a alteração no seu preço (linha `30` de `New.xml`). Considerando que o conteúdo do documento `Doc_A.xml` está armazenado em tabelas relacionais, a conversão das modificações detectadas vai refletir em duas operações de atualização do banco de dados. A primeira operação corresponde à inserção de uma nova tupla na tabela `names`; e a segunda à atualização do conteúdo da coluna `price` da tabela `book`, que mudou de `42.25` para `53.58`. O conteúdo atualizado no banco de dados está destacado nas tabelas da Figura 3.2.

names

first	last
Darcy	Gerbarg
W.	Stevens
Stephen	Rago
...	...

book

title	year	publisher	edition	price
The Economics of...	1999	Springer	1999 edition	223.48
TCP/IP Illustrated	1993	Addison-Wesley	US ed	7.82
Data on the Web...	1999	Morgan Kaufmann		75.95
Advanced Program...	2005	Addison-Wesley	2	53.58
...	...	...	...	...

Figura 3.2. Exemplo de atualização de conteúdo em tabelas relacionais.

Para solucionar o problema de manutenção de visões relacionais materializadas, optou-se por uma abordagem que utiliza detecção de diferenças e rastreamento de dados através de chaves. Na detecção de diferenças é empregado um algoritmo de *Diff*, o mesmo apresentado na seção 2.3.1. Já para a geração das chaves, usa-se uma estratégia baseada no conceito das entidades XML no esquema relacional. A partir dessa metodologia, as operações de modificação, detectadas nos documentos XML, podem ser referenciadas às tuplas nas tabelas do banco de dados através da relação entre a chave XML com a chave primária da tabela.

Assumindo que o componente XMap tenha mapeado e inserido os dados XML nas tabelas relacionais a partir do documento XML com as chaves geradas, este trabalho resume-se em atualizar o BDR à medida que os documentos XML sofrerem alterações. Por outro lado, se a alteração ocorrer no esquema dos documentos, a devida evolução do esquema XML para o relacional é assumida pelo componente SUpdater, ainda em processo de definição e fora do escopo desta dissertação. Dessa forma, a cada evolução de esquema, o SUpdater emite um aviso ao DBUpdater para que o conteúdo referente ao esquema evoluído também seja atualizado.

Em razão do DBUpdater usar o documento de mapeamento gerado por CMap e o XMap para gerar a instrução de inserção, as seções 2.1.2 e 2.1.3 apresentaram o

funcionamento destes componentes. A partir de então, a função do documento de mapeamento e do XMap no DBUpdater pode ser mais facilmente entendida.

3.2. Arquitetura Proposta

Partindo da arquitetura do *framework* X2Rel, o presente trabalho apresenta as funcionalidades de gerar as chaves para os dados XML e propagar as modificações de conteúdo dos documentos XML para o BDR. Tais funcionalidades são incorporadas ao X2Rel através dos componentes XKGen e DBUpdater, conforme ilustrado na Figura 3.3 e descrito a seguir.

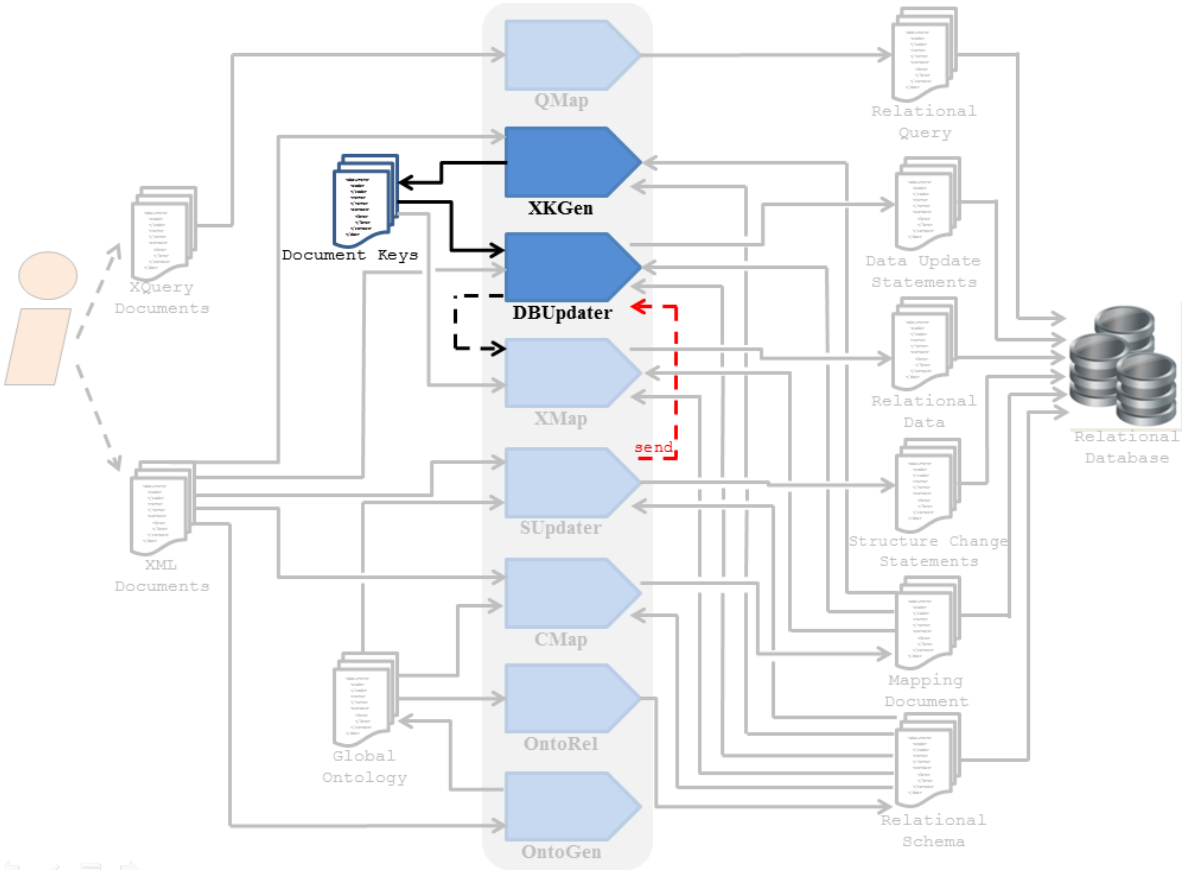


Figura 3.3. Arquitetura atualizada do *framework* X2Rel.

- XKGen (*XML Key Generator*): gera as chaves para rastrear os dados XML com as tuplas nas tabelas do BDR. Este componente recebe o documento XML, o esquema

relacional descrito em formato XML e o documento de mapeamento e produz uma representação idêntica do documento XML fonte, porém, incluindo as chaves geradas;

- DBUpdater (*Database Updater*): identifica e propaga as modificações no conteúdo dos documentos XML para operações de atualização sobre o BDR. Este componente recebe duas versões do documento XML fonte, o documento com as chaves geradas, o esquema relacional descrito em XML e o documento de mapeamento e produz um *script* com as instrução SQL DML de atualização do banco de dados. Caso a operação for de inserção, o DBUpdater envia as informações necessárias para XMap produzir as correspondentes instruções de *insert*.

  A seguir é detalhada a arquitetura dos componentes XKGen e DBUpdater.

3.2.1. Componente XKGen

O cenário ideal seria se os dados XML compartilhassem um identificador com as tuplas relacionais, o que forneceria uma condição para rastrear os dados XML armazenados no BDR. No entanto, os documentos XML podem não estar estruturados com identificadores, como é o caso das versões do documento XML da Figura 3.1 que está armazenado nas tabelas relacionais da Figura 3.2. Uma estratégia para solucionar a questão de identificadores é atribuir uma chave a cada conjunto de dados XML que correspondem a uma tupla relacional e compartilhar com as tabelas do banco de dados. Conforme exemplificado na Figura 3.4, cada subárvore `names` representa uma tupla na tabela relacional `names`.

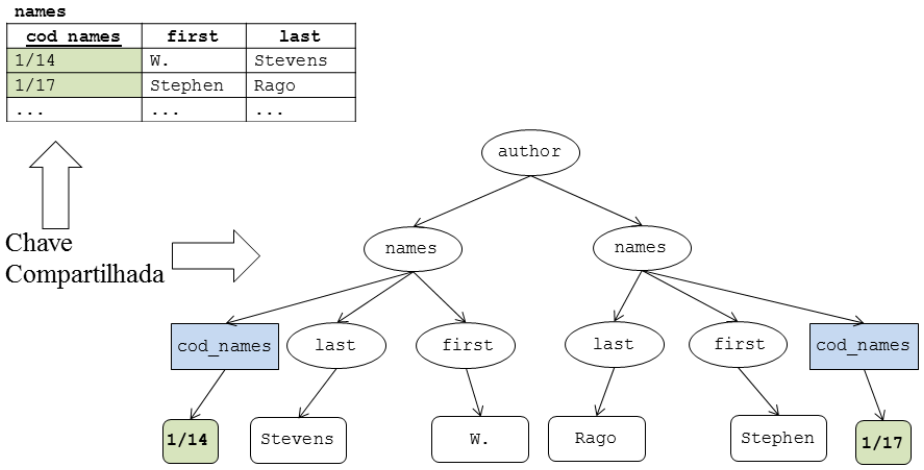


Figura 3.4. Exemplo de tupla na tabela relacional e no documento XML.

O atributo `@cod_names`, das subárvores `names`, tem a função de chave identificadora e é compartilhado na tabela `names`, ou seja, a mesma chave identifica a tupla no documento XML e na tabela relacional.

De forma a permitir o rastreamento dos dados XML armazenados em tabelas do banco de dados, tem-se a necessidade de um mecanismo que permita referenciar as entidades XML com as tuplas das tabelas relacionais. Neste sentido, foi proposto o componente XKGen que tem a funcionalidade de gerar as chaves para documentos XML de acordo com sua representação no esquema relacional correspondente. A arquitetura proposta está ilustrada na Figura 3.5.

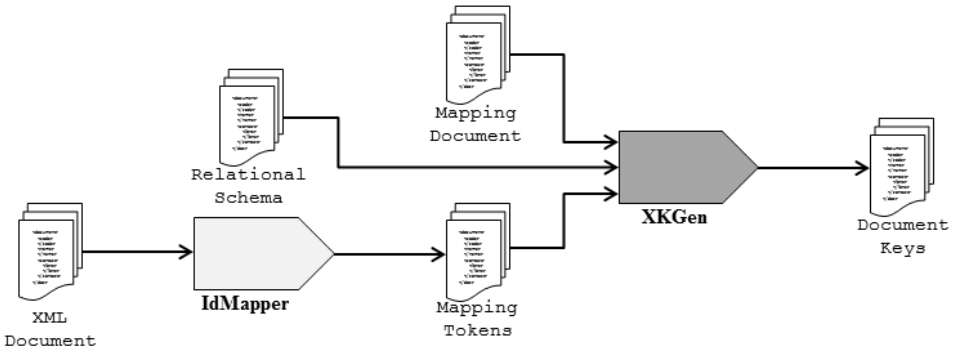


Figura 3.5. Arquitetura do componente Gerador de Chaves XML.

Conforme a arquitetura do XKGen, o primeiro passo para gerar as chaves XML é obter os artefatos de entrada: o documento de mapeamento, o esquema relacional e o documento com os identificadores e propriedades dos nodos. Este último artefato é gerado através do Mapeador de Identificadores (IdMapper), que tem como entrada um novo documento XML ou uma modificação em um documento existente e gera uma representação em XML contendo as propriedades (como exemplo: identificador, posição, nome, conteúdo e tipo de nodo) de todos os *tokens* do documento. Tal representação está sincronizada com o documento XML de origem e é atualizada automaticamente de acordo com as modificações realizadas nos dados fonte.

Os arquivos de entrada são fundamentais no processo de geração das chaves, conforme descritos a seguir:

- Esquema relacional: arquivo XML que descreve o esquema lógico relacional de banco de dados, criado pelo componente OntoRel.

- Documento de mapeamento: arquivo XML de mapeamento representando a equivalência de conceitos presentes nos documentos XML, na correspondente ontologia (gerada por OntoGen) e no esquema lógico relacional (gerado por OntoRel). Este artefato é criado pelo componente CMap;
- Documento de mapeamento dos *tokens*: arquivo XML de armazenamento dos identificadores e informações dos *tokens* do documento XML, gerado pelo IdMapper;

A partir dos artefatos de entrada, o XKGen tem a tarefa de gerar as chaves para o documento XML fonte. Para isso, os *tokens* mapeados no documento de mapeamento dos *tokens* recebem o atributo chave – previamente gerado por OntoRel – de acordo com seu conceito no esquema relacional. Como saída, tem-se o documento XML com as chaves geradas mantendo a mesma estrutura do documento de origem. O processo de geração das chaves é detalhado na seção 3.3.

3.2.2. Componente DBUpdater

De maneira a propor uma abordagem para propagar modificações de dados XML para o BDR, a arquitetura do sistema deve solucionar os principais problemas decorrentes do mapeamento de operações de edição XML para o modelo relacional. De acordo com a Figura 3.6, que ilustra a arquitetura do componente DBUpdater, a primeira etapa (detecção de deltas) corresponde à detecção do delta entre duas versões de um documento XML.

Para manter as versões, adotou-se uma estratégia onde a primeira versão representa o documento XML original (`Old`) e a segunda versão o documento modificado (`New`), conforme ilustrado na Figura 3.1. Com isso, o problema resume-se em detectar as diferenças entre duas versões de um documento e gerar o delta com as operações de edição.

A segunda etapa (atualização do banco de dados) tem a função de compor as operações de atualização sobre o banco de dados. Para isso, tem como entrada os arquivos necessários para mapear as operações de edição XML para as instruções de atualização correspondentes na linguagem SQL. Além do documento de mapeamento e esquema relacional, o processo de composição das instruções também tem como entrada o arquivo delta e o documento XML com as chaves geradas.

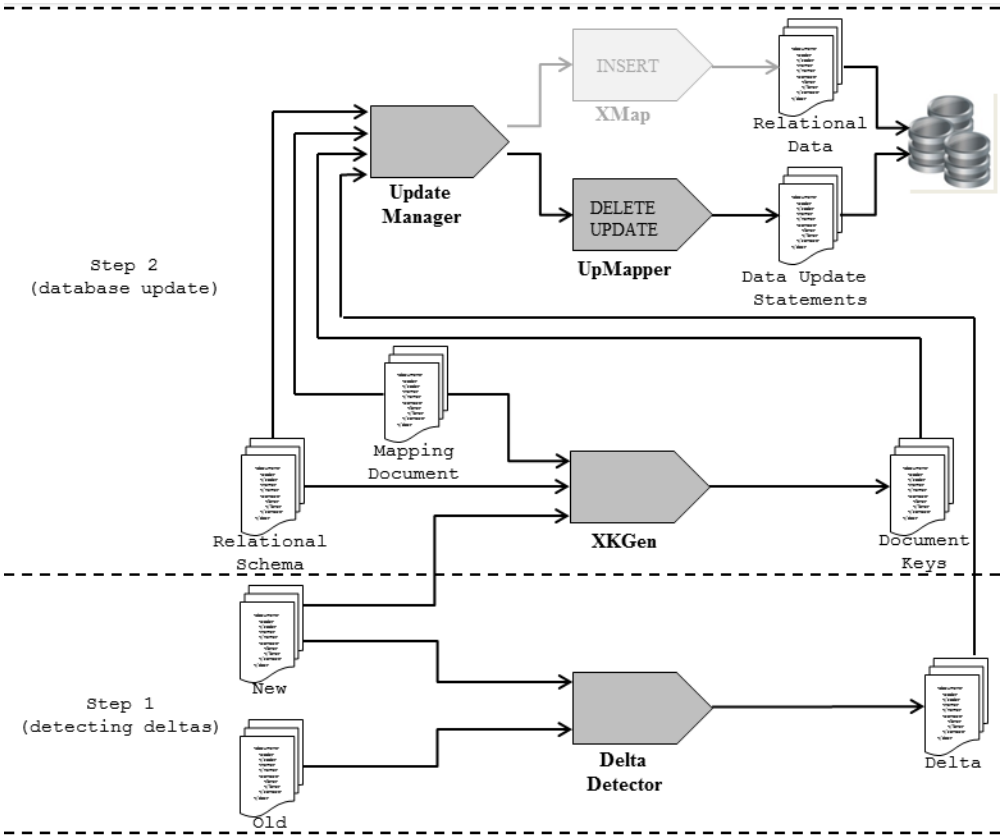


Figura 3.6. Componente DBUpdater inserido na arquitetura do *framework* X2Rel.

O núcleo central do DBUpdater é o componente Gerenciador de Atualizações (*Update Manager*), o qual tem a função de gerar as equivalências para as operações de edição no modelo XML em relação ao modelo de operações relacional, na linguagem SQL. Além disso, o gerenciador de atualizações tem a tarefa de mapear e encaminhar as informações necessárias aos componentes XMap e UpMapper (*Update Mapper*) para a composição das instruções de atualização SQL: caso for *insert*, a instrução é criada através do XMap; caso for *delete* ou *update*, a instrução é gerada pelo UpMapper.

3.3. Definição das Chaves

O processo de definição das chaves XML é feito com base na equivalência dos conceitos do documento XML no esquema lógico relacional, sendo identificado através do

documento de mapeamento gerado por CMap. Dependendo do conceito, o elemento XML receberá um atributo chave, o mesmo gerado de maneira artificial[5] na concepção das tabelas.

Para atribuir o valor ao atributo chave, leva-se em conta a unicidade da chave dentre uma coleção de documentos XML. Chaves ambíguas podem ocorrer entre um documento e outro. Além disso, é conveniente que as chaves existentes não necessitem ser atualizadas em resposta à inserções ou deleções na estrutura do documento. Desse modo, a atribuição de rótulos identificadores aos documentos e aos seus dados é uma maneira conveniente para a proposta de geração dos valores das chaves.

De maneira a gerar as chaves XML sem alterar a estrutura original do documento XML, convencionou-se uma réplica do documento XML como artefato de representação do documento estruturado com chaves. Tal representação é composta pela mesma estrutura do documento XML fonte, entretanto, incorporando as chaves geradas.

3.3.1. Atribuição dos Rótulos Identificadores

A geração e atribuição dos rótulos identificadores são realizadas pelo componente Mapeador de Identificadores, conforme ilustrado e descrito na arquitetura do componente XKGen, na seção 3.2.1. Para atribuir e manter os rótulos adotou-se o esquema de rotulagem MDL, detalhado na seção 2.4.1, que foi adaptado para as particularidades deste trabalho.

Para exemplificar a geração e manutenção dos rótulos é utilizado o fragmento da versão modificada do documento `Doc_A.xml` ilustrado na Figura 3.7.

Doc_A.xml (fragmentado)

```
 ...
 3.<book year="1999">
 4. <title>The Economics of...</title>
 5. <author>
 6.  <names>
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10.  <instituition>CITI</instituition>
11. </author>
12. <publisher>Springer</publisher>
13. <edition>1999 edition</edition>
14. <price>223.48</price>
15.</book>...
```

Figura 3.7. Exemplo de documento XML modificado.

[5] Campo chave primária artificial: é previamente gerado a partir do nome `cod_` mais o nome da tabela.

Neste exemplo, é fornecida uma visão geral para o processo de manutenção dos rótulos diante da inserção de novos *tokens* no documento XML. Além da regra geral para inserção (regra `1` de ambos os casos), também são utilizadas regras específicas para tratar a inserção de um novo nodo à direita e entre dois nodos irmãos[6]:

- Regras que tratam a inserção de um novo *token* à direita de um *token* irmão:

```
1) tokenID(instituition) ← maxTokenID(9) + 1
2) LeftID(instituition) ← TokenID(names)
3) RightID(instituition) ← Null
4) RightID(names) ← tokenID(instituition)
5) ParentID(instituition) ← ParentID(names)
```

- Regras que tratam a inserção de um novo *token* entre dois *tokens* irmãos:

```
1) tokenID(edition) ← maxTokenID (10) + 1
2) rightID(edition) ← rightID(publisher)
3) leftID(edition) ← leftID(price)
4) rightID(publisher) ← tokenID(edition)
5) leftID(price) ← tokenID(edition)
6) parentID(edition) ← parentID(price)
```

Ressalta-se que o valor da propriedade `maxTokenID`, da regra 1 de ambos os casos de inserção, corresponde ao identificador do último *token* do documento XML. Desse modo, o novo *token* inserido recebe um identificador seguido do valor do último *token*.

Conforme ilustrado na Figura 3.8, que corresponde a representação gráfica do documento `Doc_A.xml` da Figura 3.7, foram inseridos dois novos elementos na estrutura do documento.

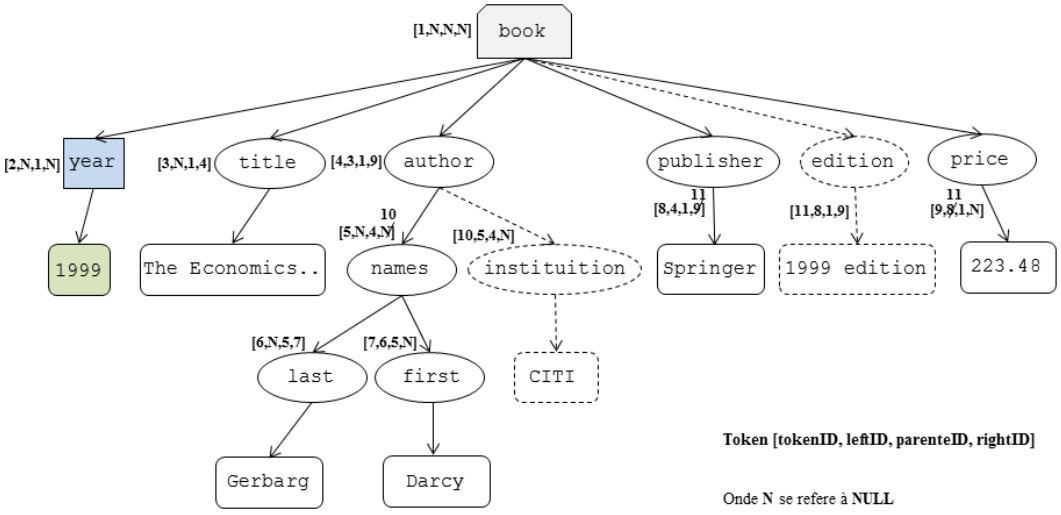


Figura 3.8. Inserção de um novo *token* na árvore XML.

[6] Os *tokens* são considerados irmãos se pertencerem ao mesmo domínio, ou seja, um elemento pode ter como irmão apenas nodos do mesmo tipo. Esse procedimento se repete para nodos do tipo atributo.

O primeiro foi `instituition` com o valor "`CITI`", que foi inserido à direita do elemento `names`, identificado como [5,N,4,10]. Tal elemento recebeu um rótulo com os identificadores [10,5,4,N]. Já no segundo caso de inserção, o elemento `edition` com o valor "`1999 edition`" é inserido entre os elementos `publisher`, identificado como [8,4,1,11], e `price`, identificado como [9,11,1,N]. Para o novo elemento é gerado e atribuído o rótulo com os identificadores [11,8,1,9].

Note que somente o identificador da relação de *tokens* irmãos foi atualizado após a inserção dos novos elementos, enquanto as relações pai-filho e ancestral-descendente permaneceram inalteradas. Ou seja, o elemento `names` passou a ter `instituition` como irmão à direita, assim como `edition` passou a ser irmão dos elementos `publisher` e `price`. Desse modo, não houve a necessidade de re-rotular os *tokens* já existentes, sendo que os mesmos permaneceram como estavam antes do processo de inserção, diferentemente dos esquemas de rotulagem apresentados em (TATARINOV et al., 2002) e (SOLTAN; RAHGOZAR, 2006), onde todos os rótulos dos *tokens* seguintes ao elemento inserido precisam ser re-rotulados.

As informações do rótulo identificador e as propriedades de cada nodo, juntamente com o identificador do documento, são mapeadas e acondicionadas em um documento XML de mapeamento dos *tokens*, conforme ilustrado na Figura 3.9.

Documento de mapeamento dos tokens (fragmentado)

```
   ...
  2.<mapping>
  3. <document id="1">                                          identificador do documento
  4.  <source name="Doc_A.xml"/>
  5.  <maxtokenid>23</maxtokenid>
  6.  <token id="1" left="" parent="" right="">
  7.   <source name="book">                                     caminho xpath do token
  8.    <xpath>/book</xpath>
  9.   </source>
 10.   <properties>
 11.    <type>element</type>
 12.   </properties>
 13.  </token>
 14.  <token id="2" left="" parent="1" right="">                 rótulo identificador do token
 15.   <source name="year">
 16.    <xpath>/book/year</xpath>
 17.   </source>
 18.   <properties>                                             Propriedades do token
 19.    <value>1999</value>
 20.    <type>attribute</type>
 21.   </properties>
 22.  </token>...
205. </document>
206.</mapping>
```

Figura 3.9. Documento de mapeamento dos *tokens*.

Cada documento XML é identificado pelo atributo `id` do elemento `<document>`, ambos na linha `3`, juntamente com o `nome` do arquivo fonte `<source>`, linha `4`, além de informar o número do último *token* (linha `5`). O rótulo identificador atribuído a cada *token* está identificado pelos atributos `id`, `left`, `parent` e `right` do elemento `<token>`, todos na linha `14`. Caso alguma das dimensões do *token* não possua relação com os demais *tokens*, o atributo que identifica a dimensão assume valor `nulo`, como em `left` e `right` na linha `14`.

Em razão das informações dos *tokens* serem de interesse no processo de identificação de equivalência no esquema lógico relacional, cada *token* é identificado pelo atributo `name` (linha `7`) e tem a expressão de caminho que o identifica na estrutura do documento apontada pela expressão `<xpath>` (linha `8`). Além disso, as propriedades do *token* que são de interesse no processo de reconstrução do documento XML são informadas no elemento `<properties>` (linha `18`).

Seguindo a relação dos identificadores nos rótulos dos *tokens* (linha `14`), segundo demonstrado em (DWEIB; AWADI; LU, 2009), o documento fonte pode ser reconstruído mantendo a mesma posição e ordem dos *tokens* da estrutura original. Isto permite que uma visão do documento fonte possa ser construída incorporando as chaves, sem a necessidade de alterar a estrutura do documento original.

O fragmento do documento de mapeamento dos *tokens* da Figura 3.9 resulta em um documento de mapeamento extenso envolvendo todos os nodos presentes no documento XML fonte. Com a finalidade de facilitar a compreensão, somente os *tokens* `1` (`book`) e `2` (`year`) foram ilustrados. O documento de mapeamento completo pode ser visto no Apêndice B.

Nesta seção foi apresentada a estratégia adotada para gerar e atribuir os rótulos identificadores aos dados XML, necessários para definir o valor das chaves. Na próxima seção será apresentado o processo de definição do atributo chave, o qual recebe um valor gerado a partir dos rótulos identificadores.

3.3.2. Definição do Atributo Chave

Para definir o atributo chave, a estratégia é identificar a equivalência do conceito de cada *token* do documento XML no esquema relacional, ou seja, se representa tabela ou

coluna. Tal processo ocorrerá de acordo com a equivalência representada no documento de mapeamento gerado por CMap. Caso for tabela, o *token* recebe o atributo chave primária, o mesmo gerado previamente por OntoRel na concepção das tabelas. Caso for coluna, o *token* é relacionado como um campo da tabela.

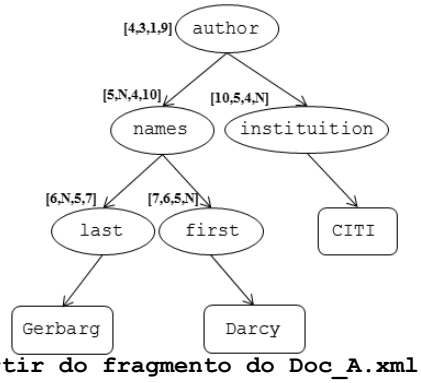
A chave XML é composta por apenas um atributo, o mesmo relacionado ao campo chave primária da tabela relacional. Tal campo é identificado no esquema lógico relacional e mapeado para uma cópia do documento XML fonte, de forma que possa identificar unicamente um conjunto de nodos alvo. Fazendo uma analogia com o esquema relacional, este conjunto de nodos seriam as colunas que compõem uma relação juntamente com seus valores. A chave tem a função de identificar exclusivamente cada um desses conjuntos que formam as tuplas das relações.

Com base nos artefatos ilustrados na Figura 3.10, em se tratando do esquema lógico relacional, a equivalência dos *tokens* `author` e `instituition` na árvore XML do documento `Doc_A.xml` correspondem à tabela e coluna, conforme o documento de mapeamento dos conceitos nas linhas 113 e 49, respectivamente. A equivalência é verificada através da comparação entre a expressão de caminho `<xpath>` do conceito no documento de mapeamento com o caminho do *token* no documento de mapeamento dos *tokens*.

Documento de mapeamento dos conceitos (fragmentado)

```
  ...
  2.<mapping>
 43. <concept name="instituition">
 44.  <source id="Doc_A.xml">
 45.   <xpath>/bibliography/book/author/instituition</xpath>
 46.  </source>
 47.  <source id="Doc_B.xml"/>
 48.  <relational>
 49.   <type>column</type>
 50.   <name>instituition</name>
 51.   <table>author</table>
 52.   <domain>string</domain>
 53.  </relational>
 54. </concept>...
105. <concept name="author">
106.  <source id="Doc_A.xml">
107.   <xpath>/bibliography/book/author</xpath>
108.  </source>
109.  <source id="Doc_B.xml">
110.   <xpath>/bibliography/book/writer</xpath>
111.  </source>
112.  <relational>
113.   <type>table</type>
114.   <name>author</name>
115.  </relational>
116. </concept>...
219.</mapping>
```

Doc_A.xml (fragmentado)



Relações geradas a partir do fragmento do Doc_A.xml

author (cod_author, instituition);
names (cod_names, last, first);
authornames (cod_author, cod_names), cod_author
references author and cod_names **references** names;

Figura 3.10. Equivalência do *token* no esquema lógico relacional.

Seja o elemento <author> (*token* id=4) do Doc_A.xml um conceito equivalente à tabela author, o campo chave primária cod_author da relação será convertido como atributo XML cod_author e mapeado como chave do elemento <author>. Da mesma forma, os elementos internos que equivalem à coluna da tabela, como instituition, são relacionados como colunas da tabela. Deve-se ressaltar que o atributo chave primária não está representado no documento de mapeamento, sendo necessário identificá-lo junto ao esquema lógico relacional.

Definido o atributo chave, a próxima seção apresenta a estratégia utilizada para gerar o valor da chave a partir dos rótulos identificadores.

3.3.3. Geração do Valor da Chave

Tendo em vista que cada documento, assim como seus *tokens*, recebe um identificador único e constante, é conveniente gerar o valor da chave a partir do valor do identificador global do *token*. Assim, as informações presentes no documento XML podem ser identificadas exclusivamente por uma determinada chave, independente do documento de origem. Além do mais, os dados XML podem ser referenciados aos dados relacionais através de um identificador comum. Por exemplo, os dados do documento XML ilustrado na Figura 3.4 e identificados pela chave XML @cod_names="1/14" podem ser rastreados na tabela names através da chave primária cod_names="1/14".

A Figura 3.11 ilustra o fragmento do documento Doc_A.xml da Figura 3.7, com as chaves geradas.

Chaves geradas para o Doc_A.xml

```
 ...
 3.<book cod_book="1/1" year="1999">
 4. <title>The Economics of...</title>
 5. <author cod_author="1/4">
 6.  <names cod_names="1/5">
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10.  <instituition>CITI</instituition>
11. </author>
12. <publisher>Springer</publisher>
13. <edition>1999 edition</edition>
14. <price>223.48</price>
15.</book>...
```

Figura 3.11. Chaves geradas para o documento A.

Tal fragmento foi reconstruído a partir das informações dos *tokens*, conforme o documento de mapeamento dos *tokens* parcialmente ilustrado na Figura 3.9, incorporando o atributo chave primária previamente mapeado do esquema lógico das tabelas. De acordo com a demonstração da Figura 3.11, o documento XML foi reconstruído mantendo a mesma relação e ordem dos *tokens* do documento XML original. Isto se deve ao relacionamento multi-dimensional entre os *tokens*, que é mantido mesmo após inserções ou deleções na estrutura do documento.

Para exemplificar, pode-se analisar a chave gerada para o elemento `<author>`, mostrado na linha 5. O valor da chave é gerado a partir da concatenação do identificador do documento, que é `1`, com o identificador global do *token*, que é `4` (vide Figura 3.8). O caractere "`/`" é usado para separar o identificador do documento do identificador do *token*. De maneira análoga, os *tokens* internos (no mesmo nível da árvore) do elemento `<author>` representam uma tupla na tabela `author`. Tal tupla é identificada unicamente pela chave primária `cod_author` com o valor "`1/4`". O mesmo processo se repete para o elemento `<names>` e seus respectivos *tokens* internos.

Conforme mencionado anteriormente, o elemento `<author>` corresponde à tabela `author` no BDR, assim como sua correspondente coluna que é o elemento `<instituition>`. Desse modo, os dados do `Doc_A.xml` são mapeados e armazenados na tabela `author`. Caso ocorra uma modificação no conteúdo do documento, a devida propagação das atualizações para os dados relacionais é feita a partir do relacionamento da chave XML com a chave primária da tabela. Com isso, a chave identifica as tuplas no documento XML e na tabela relacional.

Geradas as chaves para os documentos XML, a correspondente ontologia descritiva das instâncias XML deve ser atualizada para incorporar tais atributos à sua descrição. Consequentemente, o documento de mapeamento dos conceitos, que é gerado a partir da associação das informações presentes nos documentos XML, na ontologia e no esquema relacional também será atualizado. Este processo de atualização deve ser realizado pelo usuário do *framework* X2Rel, de maneira a reconstruir os artefatos anteriormente citados a partir dos documentos XML com as chaves geradas.

Dessa forma, torna-se possível identificar junto ao documento de mapeamento os conceitos gerados para os atributos chave da mesma forma que para as demais instâncias XML. Com isso, o conteúdo inserido no banco de dados pode usar o documento de mapeamento para endereçar o conteúdo e as respectivas chaves para as relações.

Entretanto, ressalta-se que o processo de geração das chaves limita-se a restrição de chave primária, não sendo tratada a restrição de chave estrangeira.

3.3.4. Algoritmo para Gerar as Chaves

Como ilustrado na Figura 3.9, cada documento XML é identificado pelo atributo `id` do elemento `documento` (linha 3), o qual é gerado para cada `fonte` identificada pelo atributo `name` (linha 4), segundo especificado no documento de mapeamento dos *tokens*. O elemento *token* contém o rótulo identificador do *token*, que é formado pelos atributos `id`, `left`, `parent` e `right` (linha 14), todos usados para manter as relações pai-filho, ancestral-descendente e de *tokens* irmãos. A partir disso, o atributo `name` do elemento `<source>` (linha 7) refere-se ao nome do *token*, assim como o elemento `<xpath>` (linha 8) que indica o caminho XPath que aponta para a localização do *token* no documento.

A verificação da equivalência do *token* no esquema lógico relacional é realizada a partir do documento de mapeamento dos conceitos, como descrito na seção 3.3.2 e exemplificado na Figura 3.10. Cada fonte de dados XML mapeada está identificada pelo atributo `id` do elemento `source` (linha 106), que corresponde ao nome do arquivo. A partir disso, o elemento `<xpath>` (linha 107) contém o caminho XPath que aponta para o conteúdo localizado no identificador `id`, que é de interesse para identificar a equivalência do `token`.

Em razão do esquema lógico relacional, previamente gerado por OntoRel, ter definido a coluna chave primária para a tabela, é conveniente mapear tal coluna como chave para os *tokens* que representam tabelas relacionais. Por exemplo, o conceito do elemento `<author>` (linha `5` do documento XML da Figura 3.7) verificado junto ao documento de mapeamento (linha 113 do documento de mapeamento da Figura 3.10) corresponde a tabela `author` no banco de dados, portanto recebe como chave o atributo `cod_author` (conforme esquema de tabelas da Figura 3.10). A partir disso, o valor da chave é gerado tendo em vista a unicidade da chave, ou seja, a chave deve ser única dentre uma coleção de documentos XML. Neste sentido, gerar o valor para chave a partir do identificador global do documento e do *token* mostra-se uma alternativa consistente.

O Algoritmo 1 tem como principal contribuição a geração das chaves para os documentos XML. Sua tarefa é identificar os *tokens* não-léxicos (tabelas) junto ao documento

de mapeamento dos conceitos e mapear o atributo chave primária – da correspondente tabela no esquema lógico relacional – como atributo chave no documento XML. Para isso, o laço mais externo (linha 1) percorre o documento de mapeamento dos *tokens* buscando o conceito de cada *token*, o qual é armazenado na variável da linha 3. A partir disso, o segundo laço (linha 4) percorre o documento de mapeamento e verifica o nome da tabela correspondente ao conceito não-léxico (linha 5).

Algoritmo 1: Gerar chaves para o documento XML

Entrada: DMT {documento de mapeamento dos *tokens*}
CMap {documento de mapeamento dos conceitos}
OntoRel {representação do esquema lógico relacional}
Saída: Documento com as chaves geradas

```
1. for all token em DMT do
2.   search  token → name
3.    conceito ← token
4.   for all <conceito> em CMap do
5.    if conceito = nao-lexico then
6.     search coluna → restricao PK em OntoRel
7.      atributo ← coluna + "="
8.     search id → documento em DMT
9.      chave ← atributo
10.     chave ← concat(id do documento, '/', id do token)
11.   end if
12.  end for
13. end for
```

Se o *token* for conceitualmente equivalente a tabela, então o valor do `id` do documento é concatenado com o `id` do *token*, separados pelo caractere "/", e o resultado é armazenado na variável da linha 10. Se a condição não for satisfeita, então retorna ao laço de repetição da linha 4 e repete o procedimento para o próximo conceito, até que a condição seja verdadeira. O processo se repete até que todos os *tokens* sejam verificados e, para os equivalentes a tabela, as chaves correspondentes sejam atribuídas ao documento XML.

Uma vez que as chaves foram geradas, a próxima seção detalha a detecção de deltas, necessária para identificar as operações de atualização do banco de dados.

3.4. Detecção de Deltas

A detecção de deltas é realizada pelo componente *Delta Detector*, conforme ilustrado na arquitetura do DBUpdater na Figura 3.6. Tal componente é responsável por identificar as

diferenças entre duas versões de um mesmo documento XML. O delta resultante entre a comparação destas duas versões será usado como base para, enfim, atualizar o BDR.

Para gerar o delta, o Detector de Deltas utiliza um algoritmo de *Diff* já existente. Tal algoritmo recebe a versão original e modificada do documento XML e gera as operações de edição. O Algoritmo usado é o X-Diff (WANG; DEWITT; CAI, 2003), principalmente por utilizar um modelo de operações simplificado, que é semelhante ao modelo de operações da linguagem SQL DML, e por não considerar a ordenação entre as árvores XML.

Assim como na linguagem de atualização SQL DML, o modelo de operações do X-Diff suporta inserções, exclusões e atualizações. Especificamente, as inserções e exclusões são detectadas sobre subárvores e nodos folha. Já a operação de atualização opera somente sobre nodos do tipo texto. Caso o nome de elementos ou atributos seja modificado, o algoritmo detecta como uma operação de exclusão seguida de uma inclusão. Maiores detalhes sobre o modelo de operações do X-Diff podem ser obtidas na seção 2.3.1.

Para exemplificar o processo de geração do delta, a Figura 3.13 ilustra o resultado obtido pelo X-Diff a partir da comparação entre os documentos `Old.xml` e `New.xml`, que correspondem às versões do documento XML da Figura 3.12 (réplica integral da Figura 1.2).

Documento Old.xml (fragmentado)

```
 ...
 3.<book year="1999">
 4. <title>The Economics of...</title>
 5. <author>
 6.  <names>
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10. </author>
11. <publisher>Springer</publisher>
12. <price>129.00</price>
13.</book>
```

Documento New.xml (fragmentado)

```
 ...
 3.<book year="1999">
 4. <title>The Economics of...</title>
 5. <author>
 6.  <names>
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10.  <instituition>CITI</instituition>
11. </author>
12. <publisher>Springer</publisher>
13. <edition>1999 edition</edition>
14. <price>223.48</price>
15.</book>
```

Figura 3.12. Representação da versão original e modificada do documento XML.

Conforme o delta representado na Figura 3.13, as modificações são detectadas e informadas em nodos do tipo instrução de processamento (`<? ?>`) informando o tipo de operação e o nodo sobre o qual a operação foi detectada. No exemplo, o delta está informando três operações de edição: a primeira é a inserção do nodo `<instituition>` (linha 10), com o conteúdo "`CITI`"; a segunda operação também diz respeito a uma nova inserção, dessa vez, do nodo `<edition>` (linha 13) com o valor "`1999 edition`"; a terceira e

última operação é a modificação do conteúdo no nodo <price> (linha 14), que mudou de "129.00" para "223.48". O arquivo delta completo pode ser obtido no Apêndice D.

```
...
 3.<book year="1999">
 4. <title>The Economics of...</title>
 5. <author>
 6.  <names>
 7.   <last>Gerbarg</last>
 8.   <first>Darcy</first>
 9.  </names>
10.  <instituition><?INSERT instituition?>CITI</instituition>
11. </author>
12. <publisher>Springer</publisher>
13. <edition><?INSERT edition?>1999 edition</edition>
14. <price>223.48<?UPDATE FROM "129.00"?></price>
15.</book>...
```

Figura 3.13. Delta gerado pelo X-Diff.

Uma vez que o delta foi gerado, a próxima seção detalha a estratégia usada para identificar e gerar as equivalências entre as operações detectadas no delta em relação ao modelo de operações de atualização da linguagem SQL.

3.5. Equivalência entre os Modelos de Operações

Os modelos de operações representam as características e ações de um conjunto de operações definidas para um determinado domínio de aplicação. Mais especificamente, o modelo descreve o tipo de informação que deve ser fornecida a uma determinada operação bem como o resultado obtido com a execução da mesma. Em modelos de operações para atualização de dados, cada operação é constituída por parâmetros de entrada e saída, onde são definidas propriedades e pré-condições para os dados.

Este trabalho foca na equivalência de operações entre dois modelos específicos para atualização de dados, que é o modelo de operações do algoritmo X-Diff e o modelo de operações de atualização da linguagem SQL. Ambos os modelos suportam as operações básicas, que são *insert*, *delete* e *update*, entretanto, cada um operando sobre estruturas de dados distintas (XML e relacional). Em se tratando da tradução de operações entre os dois modelos, uma determinada operação expressa num modelo pode ter um significado diferente em outro, o que torna necessário tratar essa diferença de forma que possam ser equivalentes.

Na arquitetura do DBUpdater, a tarefa de identificação das equivalências é realizada pelo componente *Update Manager*. Tal componente é responsável por tratar as operações de modificação detectadas no delta de acordo com sua equivalência na linguagem SQL. Para isso, são definidas cinco equivalências com base no modelo de operações do X-Diff (WANG; DEWITT; CAI, 2003). A Tabela 3.1 apresenta as possíveis operações correspondentes entre os dois modelos.

Tabela 3.1. Equivalência entre operações de atualização.

Modelo XML	Modelo Relacional
1. Inserção de nodo folha	Update no conteúdo de um campo da tupla
2. Exclusão de nodo folha	Update no conteúdo de um campo da tupla
3. Modificação de nodo texto	Update no conteúdo de um campo da tupla
4. Inserção de subárvore	Inserção de uma ou mais tuplas
5. Exclusão de subárvore	Exclusão de uma ou mais tuplas

Nota-se que as operações realizadas sobre nodos folha equivalem à atualização de conteúdo no modelo relacional, enquanto inserções ou deleções de subárvores correspondem ao *insert* ou *delete* de tupla. A seguir é detalhado o uso das regras de equivalência da Tabela 3.1 e o algoritmo proposto para gerar as equivalências.

3.5.1. Operações sobre Nodo Folha

Na estrutura do documento XML, nodos folha descendentes do mesmo nodo pai formam uma mesma tupla na tabela relacional. Considerando que podem ocorrer inserções ou deleções tanto de subárvores (que contém a chave da tupla) quanto de nodos folha (que representam campos da tupla), torna-se necessário tratar o tipo de operação de acordo com sua equivalência no modelo de operações de atualização da linguagem SQL.

Para exemplificar o processo de identificação da equivalência, consideramos a inserção ou exclusão de um nodo folha (equivalência 1 e 2 da Tabela 3.1) em um documento XML e sua devida tradução para os dados armazenados na base relacional. Tal operação afetará somente um campo da tupla na tabela, e, portanto é tratada como uma operação de atualização de conteúdo dos dados relacionais. Formalmente pode-se escrever:

Definição 3.1. *Seja* T *uma subárvore em um determinado documento XML. Seja* F_n *um conjunto de nodos folha descendentes de* T*. Se algum* F *for adicionado ou removido de* T*, então a operação* op *é equivalente a um update no modelo de operações de atualização da linguagem SQL.*

Conforme ilustrado na Figura 3.14, o elemento <last> foi deletado da subárvore identificada pela chave cod_names="1/5".

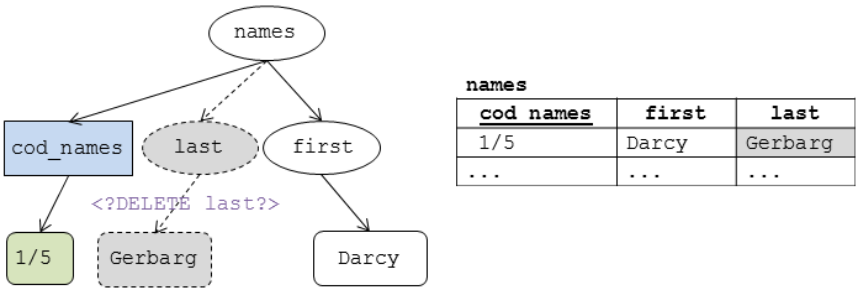


Figura 3.14. Equivalência de operação de atualização entre o modelo do X-Diff e SQL.

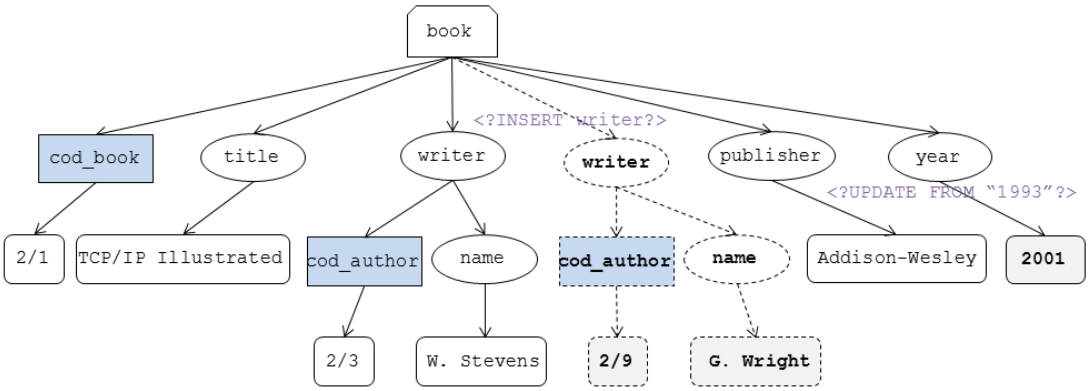
Com base na equivalência 2 (Tabela 3.1), a deleção do elemento <last> é equivalente a uma operação de atualização sobre a coluna last da tabela names, onde o campo da tupla passa a ter valor nulo (quando não há valor).

3.5.2. Operações sobre Subárvores

As subárvores contidas no documento XML compõem as tuplas das tabelas na base de dados relacional. Uma subárvore pode conter outras subárvores como descendentes, sendo que cada uma possui uma chave para identificar seu conteúdo e referencia-la com a correspondente tupla da tabela. Assim como apresentado nas equivalências 4 e 5 da Tabela 3.1, a inclusão ou exclusão de subárvores no documento XML equivale ao *insert* ou *delete* de uma ou mais tuplas no banco de dados. Formalmente, pode-se escrever:

Definição 3.2. *Seja* D *um documento XML dentre uma coleção de documentos. Seja* T_n *um conjunto de subárvores em* D*. Se qualquer* T *em* D *for adicionado ou removido, então a operação* op *é equivalente ao insert ou delete de tupla(s) na tabela relacional.*

Para exemplificar a inserção de uma nova tupla, consideramos a subárvore destacada e identificada pela chave `cod_author="2/9"` na Figura 3.15. A inserção de tal subárvore corresponde à tupla hachurada na tabela relacional `author`, que compartilha a mesma chave da tupla no documento XML. Nota-se que os elementos `<writer>` estão semanticamente referenciados aos elementos de `<book>` através da chave `cod_author`. Cada subárvore `writer` representa uma tupla na base de dados relacional. Os nodos `cod_author` e `name` da subárvore `writer` são equivalente à colunas da tabela `author`, que pode ter outras colunas, como `instituition`, oriunda dos demais documentos XML.



author

cod_author	instituition	name	cod_names
1/4	CITI		1/5
2/3	W.Stevens		
2/9		**G. Wright**	
...	...		

Figura 3.15. Exemplo de inserção de uma tupla no documento XML e na tabela relacional.

Em se tratando de operações detectadas sobre uma única subárvore, a tradução para a instrução de atualização no banco de dados tem como alvo apenas uma tupla, o que torna o processo relativamente mais simples. No entanto, inserções e exclusões podem ocorrer em subárvores que contém outras subárvores descendentes. Nesse caso, os dados alvo representam múltiplas tuplas, o que torna o processo de identificação da equivalência mais complexo em relação às demais operações.

3.5.3. Operação sobre Nodo Texto

No documento XML, elementos e atributos englobam valores de texto. Tais valores representam o conteúdo de dados armazenado nas tabelas relacionais. Portanto, modificações

nos valores dos elementos e atributos correspondem à atualizações sobre o banco de dados. Assim como expresso na equivalência 3 da Tabela 3.1, uma modificação detectada em nodos do tipo texto equivale a uma atualização de conteúdo de um campo em uma tupla. Logo, a exclusão de um nodo texto representa uma atualização sobre os dados relacionais, de modo que o campo correspondente assuma valor nulo. Formalmente, pode-se escrever:

Definição 3.3. *Seja* F_n *uma sequência de nodos folha (elementos e atributos) em um documento XML. Seja* `text` *um valor em um determinado* F_n*. Se algum* `text` *sofrer modificação em* `F`*, então a operação* `op` *representa um update no modelo de operações de atualização da linguagem SQL.*

Um exemplo de tal caso pode ser visto na Figura 3.15. O elemento <year> (ano de publicação do livro) é modificado de `1993` para `2001`. Essa operação irá refletir em uma atualização no conteúdo do campo `year` da tabela `book`, conforme esquema lógico relacional previamente gerado.

Note que em todas as equivalências de operações da Tabela 3.1, as chaves são usadas para rastrear os dados XML armazenados no banco de dados. Por exemplo, é necessário saber que o conteúdo do elemento <year> compartilha a mesma chave que a coluna `year` da tabela `book`.

3.5.4. Algoritmo para Identificar a Equivalência entre Operações

A principal contribuição do Algoritmo 2 é gerar a equivalência em SQL para as operações de edição detectadas no arquivo delta. Para isso, recebe o delta gerado pelo X-Diff e o documento de mapeamento gerado por CMap e retorna um vetor com as propriedades das operações de atualização correspondentes no formato SQL. Basicamente o algoritmo percorre o delta, que é um arquivo XML, buscando as operações de atualização. Para cada operação é gerada a equivalência correspondente, de acordo as equivalências da Tabela 3.1.

Para um melhor entendimento da proposta do Algoritmo 2, considera-se como exemplo uma operação de inserção ou deleção detectada sobre um nodo folha de um documento XML e sua devida conversão para instruções de atualização no BDR, em linguagem SQL. Nesse caso, o nodo folha representa uma coluna no esquema relacional, e,

portanto afetará somente o conteúdo de um campo da tupla na tabela. Logo, o algoritmo deve entender que tal operação equivale a uma instrução de *update* no BDR. Nas seções 3.5.1, 3.5.2 e 3.5.3 foram detalhados exemplos de equivalência entre operações no modelo XML em relação ao modelo relacional, de acordo com o contexto em que este trabalho esta inserido.

Algoritmo 2: Gerar a equivalência entre operações

Entrada: `delta` {arquivo delta gerado pelo algoritmo X-Diff}
`CMap` {documento de mapeamento}
Saída: Vetor com às operações correspondentes em SQL

```
1. for each operacao em delta do
2.   n ← nodo que contém à operação
3.   if operacao = "UPDATE" then
4.     posição ← posição de n{posição de n na árvore XML}
5.     tipo ← UPDATE
6.     conteudo ← nodo texto de n
7.     operacao ← tipo e conteudo
8.     Adiciona posição e operacao à vetor de operações
9.   end if
10.  if operacao = "INSERT" then
11.    posição ← posição de n
12.    search conceito em CMap
13.      if conceito = nao-lexico then
14.        tipo ← INSERT
15.        conteudo ← sub-árvore contida em n
16.        operacao ← tipo e conteudo
17.        Adiciona posição e operacao à vetor de operações
18.      else
19.        tipo ← UPDATE
20.        conteudo ← nodo texto de n
21.        operacao ← tipo e conteudo
22.        Adiciona posição e operacao à vetor de operações
23.      end if
24.  end if
25.  if operacao = "DELETE" then
26.    posição ← posição de n
27.    conteudo ← conjunto vazio
28.    search conceito em CMap do
29.      if conceito = nao-lexico then
30.        tipo ← DELETE
31.        operacao ← tipo e conteudo
32.        Adiciona posição e operacao à vetor de operações
33.      else
34.        tipo ← UPDATE
35.        operacao ← tipo e conteudo
36.        Adiciona posição e operacao à vetor de operações
37.      end if
38.  end if
39. end for
```

Analisando a estrutura do Algoritmo 2, o laço de repetição (linha `1`) percorre o arquivo delta e para cada operação informada entre "`<? ?>`" armazena o nodo que contém a

instrução na variável `n` (linha `2`). Para cada operação correspondente são extraídas as informações de `tipo` de operação, `conteúdo` e a `posição` do `n` no documento XML. A posição corresponde ao caminho para navegação na árvore XML e é usada, posteriormente, nas etapas de resolução de conflitos e composição das instruções, para identificar a correspondente chave da operação junto ao documento XML com as chaves geradas.

Com base nas equivalências definidas no decorrer desta seção, caso a operação informada no delta for do tipo `UPDATE` (linha 3), então sua correspondente no banco de dados é a instrução de atualização, conforme equivalência `3` da Tabela 3.1. No entanto, se a operação for do tipo `INSERT` ou `DELETE`, a instrução equivalente pode corresponder à atualização de um campo da tupla (equivalências `1` e `2`) ou à inserção ou deleção de um registro da relação, de acordo com as equivalências `4` e `5`.

Considerando que as operações de inserção e deleção podem ser detectadas tanto em nodos folha como em subárvores, torna-se necessário tratar cada tipo de operação de maneira que possam ser traduzidas para instruções de atualização sobre o banco de dados. Para isso, o documento de mapeamento é utilizado para verificar se a operação será executada sobre tabela ou coluna no banco de dados. Se for referente à tabela (linhas `13` e `19`), então significa que a operação será executada sobre a tupla de uma tabela, podendo ser `INSERT` (linha `14`) ou `DELETE` (linha `30`); senão, a operação será executada sobre determinado campo da tupla, sendo equivalente à `UPDATE` (linha `19` e `34`).

Com base no processo de geração da equivalência para a operação de inserção, o teste condicional da linha `13` verifica se a operação foi detectada sobre um conceito equivalente à tabela, caso a condição for verdadeira, então o tipo e conteúdo da operação são armazenados na variável `operacao` (linha `16`), a qual é adicionada juntamente com a `posicao` ao vetor de operações (linha `17`). Senão é adicionada como uma operação de `UPDATE` ao vetor de operações (linha `22`). O processo para tratar a operação de deleção segue a mesma lógica, no entanto o conteúdo recebe conjunto vazio (linha `27`), quando não há valor.

Mesmo identificando a equivalência de operações no modelo XML em relação ao relacional, conflitos podem ocorrer na tradução das operações entre um modelo e outro. Neste sentido, a próxima seção apresenta o processo de tratamento de conflitos, necessário para gerar as instruções de atualização no banco de dados.

3.6. Tratamento de Conflitos

O processo de propagação de modificações de dados XML para o BDR utiliza os modelos de operações de atualização do X-Diff e da linguagem SQL para identificar a equivalência entre as operações. Conflitos podem ocorrer para traduzir as operações detectadas no delta para atualizações equivalentes no banco de dados. Diante disso, torna-se necessário identificar e tratar os conflitos existentes conforme a proposta de solução em particular.

Esta seção descreve os possíveis conflitos que podem ocorrer no processo de atualização do banco de dados que armazena dados XML, os quais são tratados de acordo com a proposta dos algoritmos apresentados na seção 3.7. Detalhes de solução para os conflitos decorrentes da inserção podem ser obtidos em (AVELAR; SACCOL; PIVETA, 2012).

3.6.1. Conflitos de Deleção

Uma possibilidade de conflito é decorrente da deleção de subárvores que contém outras subárvores descendentes, o que reflete na deleção de múltiplas tuplas do BDR. Neste caso, a operação informada no arquivo delta deve ser traduzida para tantas instruções de deleção em SQL quantas forem necessárias.

Em razão do documento XML com as chaves geradas ser atualizado de maneira independente do delta, optou-se por manter temporariamente uma versão não atualizada (original) deste documento. Isto se justifica para o caso específico de tradução da operação de deleção para a instrução correspondente no banco de dados, pois uma vez deletada a chave torna-se impossível relacionar o conteúdo da operação com a chave no documento, a qual está relacionada a chave da correspondente tupla no BDR.

Para solucionar o conflito de deleção de múltiplas tuplas, o DBUpdater usa a estratégia de identificar todas as chaves contidas no conteúdo deletado. Para isso, o Gerenciador de Atualizações obtém a posição da operação no delta e a associa na versão não atualizada (original) do documento com as chaves geradas, para identificar o conteúdo deletado.

Considerando que a deleção pode ser referente a mais de uma tupla, é verificado junto ao documento XML do esquema lógico relacional os nodos contidos no conteúdo deletado que correspondem à restrição do tipo chave primária das tabelas. Com isso, obtém-se a chave primária que determina a condição da instrução.

Partindo do exemplo da Figura 3.16, a deleção do elemento <author>, em destaque, corresponde à remoção de uma subárvore do documento Doc_A.xml. Note que o elemento deletado possui dentre seus descendentes a subárvore <names> (linha 6 do Doc_A.xml), que por sua vez corresponde a uma tupla da tabela names. Portanto, a propagação desta deleção para o banco de dados deve refletir tanto na tabela author quanto na tabela names, deletando as tuplas cujo conteúdo está identificado pelas chaves cod_author="1/4" e cod_names="1/5", conforme linhas 5 e 6 (respectivamente) do documento A.

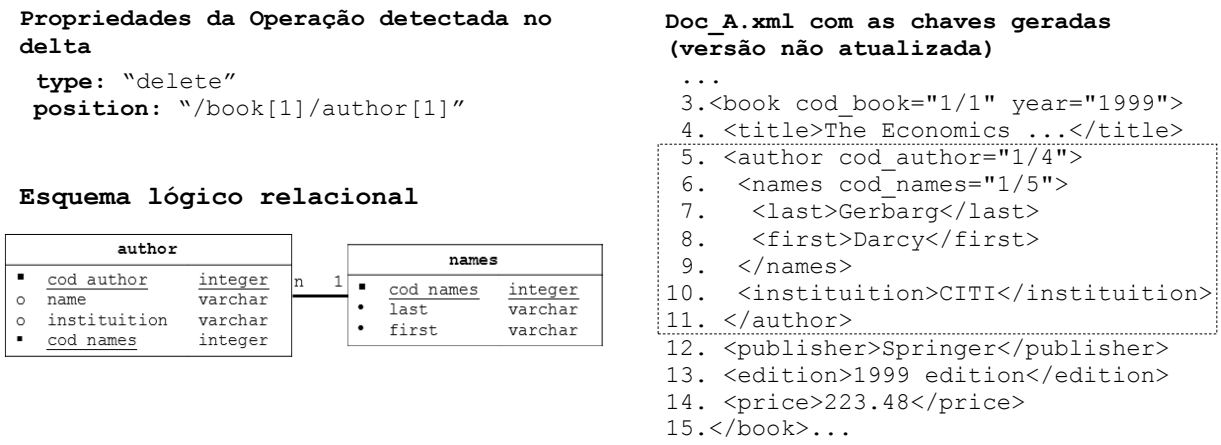


Figura 3.16. Propriedades da operação, documento XML com as chaves geradas e o correspondente esquema das tabelas relacionais.

Analisando a Figura 3.16, a posição correspondente à operação detectada no delta, associada ao Doc_A.xml, revela que a deleção corresponde ao elemento <author> (linha 5) e seus nodos descendentes. No exemplo, os elementos <author> e <names> correspondem a tabelas relacionais, portanto, ambos possuem chaves identificadoras que permitem rastrear os dados XML nas tabelas relacionais. O campo chave primária das tabelas author e names são buscadas no esquema lógico relacional, que neste caso, corresponde aos campos cod_author e cod_names. A partir disso, para cada deleção, o nome da tabela e a correspondente chave que referencia a tupla são encaminhadas ao componente Mapeador de Atualizações, o qual é responsável por montar a instrução de deleção correspondente seguindo a cláusula SQL.

3.6.2. Conflitos de Inserção

Os principais conflitos que podem ocorrer na inserção dos dados XML no BDR são tratados em tempo de execução por XMap. São tratados conflitos decorrentes de nomenclatura, estrutura, elemento inexistente e representação, conforme detalhado em (AVELAR; SACCOL, PIVETA, 2012). Para exemplificar, a Figura 3.15 ilustra graficamente a subárvore, identificada pelo elemento `<writer>`, inserida em `Doc_B.xml`. Tal inserção refletirá na tabela `author` representada no esquema lógico da Figura 3.16. O documento de mapeamento usado para solucionar possíveis conflitos de inserção é referente à representação do Apêndice `A`.

Para associar os dados modificados com as chaves geradas para o correspondente documento XML, uma busca é realizada no delta onde para cada operação é obtido o seu tipo e posição no arquivo delta. Para detectar a equivalência da operação no modelo da linguagem SQL, são seguidas as regras de equivalência definidas na seção 3.5. A partir da correspondente operação, a posição do nodo identificado pela operação é associada ao documento com as chaves geradas – versão atualizada. Com isso, identifica-se a chave que referencia os dados da operação. Os dados relacionais referentes à tabelas e colunas são identificados junto ao documento de mapeamento gerado por CMap e utilizados para compor o cabeçalho da instrução.

Partindo do exemplo, há pelo menos dois conflitos a serem tratados no mapeamento dos dados XML para a tabela relacional. O primeiro deles é referente ao conflito de nomenclatura sinonímia, que ocorre quando o mesmo conceito é descrito por dois ou mais nomes. Como pode ser observado no documento de mapeamento, o nome do elemento `<writer>` de `Doc_B.xml` é sinônimo do nome do elemento `<author>` de `Doc_A.xml`, portanto, um conflito de sinonímia está presente para o conceito não-léxico.

O segundo conflito é do tipo elemento inexistente, que ocorre na ausência de dados para armazenamento em banco de dados. No exemplo, a tabela `author` possui os campos `instituition` e `cod_names` que é chave estrangeira da tabela `names`. Ao tentar armazenar o conteúdo do `Doc_B.xml`, haverá uma falha devido à ausência dos elementos `<instituition>` e `<cod_names>`. Como solução, os campos correspondentes aos elementos ausentes assumirão valor `null` na inserção no banco de dados.

Resolvidos os conflitos decorrentes da transformação de operações de atualização em dados XML para atualizações não BDR, na próxima seção é detalhado o processo de composição das instruções em linguagem SQL.

3.7. Composição das Instruções

O processo de composição das instruções de atualização tem o objetivo de criar as instruções de atualização em linguagem SQL, obedecendo as regras de integridade referencial do banco de dados. Para isso, são relacionados, como entrada, o esquema lógico relacional, as definições do documento de mapeamento e o conteúdo modificado nos documentos XML. Como saída, tem-se o *script* com as instruções de atualização do conteúdo XML no BDR.

A arquitetura do DBUpdater incorpora dois subcomponentes para realizar a montagem das instruções de atualização. O primeiro, denominado de Mapeador de Atualizações, é responsável pelas operações de deleção (*delete*) e atualização (*update*). Já para tratar da inserção (*insert*) de novas tuplas, é utilizado o componente XMap, apresentado em (AVELAR; SACCOL, PIVETA, 2012), o qual também desempenha a tarefa de mapear e inserir os dados originais no banco de dados.

Algoritmos foram definidos para resolver conflitos decorrentes do processo de tradução das operações de `DELETE` e `UPDATE` para o banco de dados, bem como para criar a estrutura da instrução correspondente a cada tipo de operação, conforme apresentado nas subseções seguintes. Para a operação de inserção, a tarefa de resolver os conflitos é delegada ao componente XMap.

3.7.1. Instrução de Deleção

O processo de composição da instrução de deleção ocorre a partir das operações de edição detectadas e informadas no arquivo delta. A partir disso, o delta passa por um processo de tradução para operações de atualização equivalentes em SQL, como detalhado na seção 3.5. O Algoritmo 2 tem a tarefa de identificar e gerar um vetor com as informações de cada

operação correspondente em SQL, conforme o tipo de instrução de atualização. As informações referentes às operações de deleção são usadas como base para tratar possíveis conflitos e para compor a estrutura da cláusula da instrução. A tarefa de identificação de operações equivalentes é realizada pelo componente Gerenciador de Atualizações.

Cada operação equivalente a deleção é traduzida para a estrutura de uma cláusula `DELETE` em SQL, a qual será executada sobre registros de uma tabela relacional, de acordo com a chave primária que identifica a tupla na relação. A cláusula `DELETE` em SQL segue a seguinte estrutura:

```
DELETE FROM <tabela> WHERE <condição>
```

Onde o comando `DELETE` referencia o nome da tabela sobre a qual a operação será executada e a condição que indica o que será removido. A condição é determinada pelo atributo chave primária, o qual foi previamente gerado na etapa de definição das chaves, na seção 3.3. De maneira genérica, a operação de deleção tem a função de remover registros de tabelas do BDR usando como filtro a chave primária da relação.

Conforme apresentado na seção 3.6, existem conflitos que podem ocorrer no processo de tradução da operação de deleção detectada no delta para o BDR. O documento XML com as chaves geradas e o documento XML do esquema lógico relacional servem como artefatos no processo de transformação da operação para instruções de atualização correspondentes em SQL, além de auxiliar na resolução de conflitos. O Algoritmo 3 tem a tarefa de resolver o conflito de deleção de múltiplas tuplas, o qual ocorre quando uma operação de deleção é detectada em uma subárvore que possui outras subárvores descendentes, conforme exemplificado na seção 3.6.1.

Para resolver o conflito de deleção, uma instrução de deleção deve ser gerada para cada subárvore que corresponde a uma tupla no banco de dados. Dessa forma, o conteúdo referente a tupla deletada no documento XML reflete na remoção da tupla correspondente na tabela relacional.

O Algoritmo 3 tem três artefatos de entrada, sendo eles: o vetor com as informações das operações correspondentes em SQL, gerado pelo Algoritmo 2; o documento XML com as chaves geradas, que neste caso é usada a versão não atualizada do documento; e o esquema lógico relacional descrito em formato XML, gerado por OntoRel. Como saída, tem-se um vetor com as informações de cada operação de deleção, necessárias para compor a cláusula da instrução de `DELETE` em linguagem SQL.

Algoritmo 3: Resolver o conflito de deleção de múltiplas tuplas

Entrada: `vetorOperações` {vetor com às operações correspondentes em SQL}
`chavesGeradas` {documento XML com as chaves geradas, versão não-atualizada}
`OntoRel` {representação do esquema lógico relacional}
Saída: Vetor com informações das operações de deleção

```
1. for all operação="DELETE" em vetorOperações do
2.   posição ← posição de operação
3.   arquivo ← chavesGeradas
4.   conteúdo ← ler(posição em arquivo)
5.   conteudoDeletado ← conteúdo
6.   for all tabela em OntoRel do
7.     tabela ← nome da tabela
8.    search coluna → restricao PK
9.      atributo ← coluna
10.    search atributo → conteudoDeletado
11.      chave ← atributo
12.      if atributo=chave then
13.        instrução ← tabela e chave
14.        Adiciona instrucao à vetor de operações de deleção
15.      end if
16.  end for
17. end for
```

Partindo da estrutura do Algoritmo 3, o laço de repetição mais externo percorre o vetor com as informações das operações correspondentes em SQL em busca de operações do tipo `DELETE`. Para cada operação, a posição do nodo deletado é armazenada na váriavel `posicao` e o documento XML com as chaves na variável `arquivo`, linhas `2` e `3` (respectivamente). O conteúdo deletado, referenciado pela posição da operação, é lido no documento XML com as chaves e armazenado na variável `conteudo` (linha `4`), que por sua vez é alocado à variável `conteudoDeletado` da linha `5`. Em conteúdo deletado, cada conjunto de dados referente à tupla no documento XML é identificado pela chave de rastreamento com a tupla relacional, a qual corresponde à condição da instrução de deleção.

A partir disso, a próxima etapa tem a tarefa de buscar a restrição de chave primária das tabelas presentes no esquema lógico relacional, a qual será relacionada com os nodos presentes no conteúdo deletado para identificar a correspondente chave da tupla. Para isso, o laço de repetição da linha `6` percorre o documento XML do esquema lógico relacional para obter o nome da tabela e a correspondente restrição de chave (linhas `7` e `9`). Com isso, obtém-se o nome da tabela e da chave, que é comparada com os nodos (*tokens*) contidos no conteúdo deletado, conforme a condição da linha `12`. Se a condição for verdadeira, o *token* chave e a tabela, são adicionadas à instrução (linha `13`). Por fim, a instrução é adicionada ao vetor com as informações para compor as instruções de deleção em SQL (linha `14`).

Uma vez gerado o vetor com as informações de cada operação de deleção, realizado pelo Algoritmo 3, a etapa seguinte é compor o cabeçalho da instrução seguindo a cláusula `DELETE` de SQL. Tal tarefa é realizada pelo componente Mapeador de Atualizações, cuja funcionalidade está discriminada no pseudocódigo do Algoritmo 4.

Algoritmo 4: Criar instrução de deleção
Entrada: `vetorDeleções` {vetor com informações das operações de deleção}
Saída: Instruções de deleção no banco de dados
1. **for all** `instrução` em `vetorDeleções` **do**
2. `tabela` ← `tabela` de `instrução`
3. `condição` ← `chave` de `instrução`
4. `instruçãoDeleção` ← montar("`DELETE FROM`" + <`tabela`> + "`WHERE`" + <`condição`>)
5. **end for**

De acordo com a estrutura do Algoritmo 4, responsável pela tarefa de criar e preencher o cabeçalho da instrução de `DELETE` a partir do vetor de deleções gerado pelo Algoritmo 3, o laço de repetição percorre o vetor e para cada instrução presente é armazenado o nome da `tabela` e a `condição` – que é determinada pela chave, nas variáveis das linhas `2` e `3` do Algoritmo 4. O cabeçalho da instrução (linha `4`) é montado e preenchido com os valores das variáveis `tabela` e `condição`, seguindo a estrutura da cláusula `DELETE`. Desse modo, a montagem da instrução deverá produzir a instrução de deleção em BDR referenciando de maneira correta o nome da tabela e a chave primária da condição.

3.7.2. Instrução de Atualização

A instrução de `UPDATE` tem a função de alterar dados de registros em tabelas do banco de dados. Tais registros podem ser identificados através de um filtro, que neste caso é determinado pela chave primária. Assim como na operação de `DELETE`, para recuperar a chave referente à condição da instrução, a posição do conteúdo alterado é associada com o documento XML com as chaves geradas. Além disso, também são usados como artefatos de entrada o documento de mapeamento e a representação do esquema lógico relacional, os quais são necessários para identificar as informações que preenchem as palavras reservadas na cláusula da instrução de `UPDATE`.

O cabeçalho que compõe a cláusula da instrução de UPDATE em linguagem SQL segue a seguinte estrutura:

```
UPDATE <tabela> SET <coluna=valor> WHERE <condição>
```

Considerando que o delta gerado pelo algoritmo de *Diff* detecta a operação de UPDATE sobre apenas um nodo do tipo texto, que corresponde a uma coluna na tabela relacional, sua tradução para o formato SQL tem como alvo o valor de um único campo da tupla. Para obter as informações referentes ao nome da tabela, os dados alvo e a chave que referencia a correspondente tupla, o Algoritmo 5 associa as informações presentes nos artefatos de entrada e produz como saída um vetor com as propriedades de todas as operações de UPDATE correspondentes.

Algoritmo 5: Buscar operações de atualização

Entrada: chavesGeradas {documento XML com as chaves geradas, versão atualizada}
vetorOperações {vetor com às operações detectadas no delta}
CMap {documento de mapeamento}
OntoRel {representação do esquema lógico relacional}
Saída: Vetor com as informações das operações de atualização

```
1. for all operação= "UPDATE" em vetorOperações do
2.   posição ← posição de operação
2.   delta ← conteúdo de operação
3.   arquivo ← chavesGeradas
4.   expressão ← buscar(ancestral de posição)
5.   conteúdo ← ler(posição em arquivo)
6.   fragmento← ler(expressão em arquivo)
7.   for each tabela em OntoRel do
8.     tabela ← nome da tabela
9.     search coluna → restricao PK
10.      atributo ← coluna
11.    search atributo → fragmento
12.      chave ← atributo
13.      if atributo=chave then
14.        chave ← atributo
15.      search caminho → conceito em CMap
16.        caminho ← xpath de conceito
17.        if compatível(caminho com posição) then
18.          coluna ← nome da coluna
19.          valor ← delta
20.          dadosAlvo ← coluna e valor
21.          Adiciona tabela, dadosAlvo e chave à vetor de operações de atualização
22.      end if
23.  end for
24. end for
```

O processo de identificação das propriedades da operação é semelhante à rotina implementada no Algoritmo 4, porém, o Algoritmo 5 busca informações específicas para

compor a cláusula de UPDATE, incluindo o conteúdo alterado. De acordo com a estrutura do algoritmo, o laço de repetição mais externo (linha 1) percorre o vetor de operações, gerado pelo Algoritmo 2, em busca de operações de UPDATE. A condição da linha 2 verifica se a operação é do tipo UPDATE e realizada uma busca pelo ancestral do nodo que contém a operação. O ancestral do nodo que contém a operação é armazenado na váriavel expressão (linha 4), que por sua vez é lida no documento XML com as chaves geradas (linha 6) e armazenado na váriavel fragmento. Da mesma forma, a posição da operação também é lida no documento (linha 5). Com isso, obtém-se a subárvore correspondente ao conteúdo atualizado, a qual contém a chave e os dados alvo para a atualização.

O nome da tabela e a correspondente coluna referente à restrição de chave primária são buscadas junto ao esquema relacional e alocadas às variáveis das linhas 8 e 10. Com isso, têm-se as informações necessárias para identificar junto ao conteúdo modificado e ao documento de mapeamento, a tabela, os dados alvo e a respectiva chave para compor a instrução. A chave é identificada através da condição da linha 13, onde os *tokens* contidos na subárvore ancestral do nodo informado na operação são comparados com o nome da chave, retornando, caso a condição for satisfeita, o *token* correspondente à chave (linha 14).

A etapa seguinte é verificar a compatibilidade do caminho do conceito com a posição do nodo que contém a operação. Se as expressões de caminho forem compatíveis, então a coluna e seu respectivo valor são adicionados à variável dadosAlvo (linha 20). A partir disso, a tabela, os dados modificados e a chave são adicionados ao vetor de operações de UPDATE (linha 21), de maneira que possam ser facilmente manipuladas na etapa de composição das instruções.

Gerado o vetor com as propriedades necessárias para preencher o cabeçalho das instruções, realizado pelo Algoritmo 5, a próxima etapa é montar as instruções de atualização seguindo a cláusula UPDATE da linguagem SQL. Para realizar esta tarefa, o componente Mapeador de Atualizações apresenta o Algoritmo 6, cuja estrutura segue a mesma rotina do Algoritmo 4, onde o laço de repetição percorre o vetor com as instruções de atualização e obtém as propriedades referentes à tabela, à coluna e seu respectivo valor e à condição, que é determinada pela chave, conforme variáveis das linhas 2–4.

Para cada instrução correspondente é criado um cabeçalho de acordo com a estrutura da cláusula UPDATE em SQL, o qual é preenchido com os valores das variáveis correspondentes.

Algoritmo 6: Criar instrução de atualização

Entrada: `vetorAtualizações` {vetor com informações das operações de atualização}
Saída: Instruções de atualização de conteúdo no banco de dados

1. **for each** `instrução` em `vetorAtualizações` **do**
2. `tabela` ← `tabela` de `instrução`
3. `coluna` ← `dadosAlvo` de `instrução`
4. `condição` ← `chave` de `instrução`
5. `instruçãoAtualização` ← montar("`UPDATE`" + <`tabela`> + "`SET`" + <`coluna=valor`> + "`WHERE`" + <`condição`>)
6. **end for**

Com isso, a criação da instrução (linha `5`) deverá produzir a instrução de `UPDATE` para ser executada em BDR de maneira que traduza corretamente as modificações nos dados XML para os dados relacionais correspondentes.

3.7.3. Instrução de Inserção

As operações do tipo inserção informadas no arquivo delta devem ser traduzidas para instruções de atualização sobre o banco de dados na linguagem SQL. Para realizar essa tarefa, o DBUpdater incorporou à sua arquitetura o componente XMap, apresentado em (AVELAR; SACCOL, PIVETA, 2012) e brevemente detalhado na seção 2.1.3. Entretanto, antes de tudo é necessário obter o conteúdo referente à operação para então encaminhar ao XMap. Para isso, o Gerenciador de Atualizações do DBUpdater usa o Algoritmo 7, o qual tem a tarefa de buscar as operações de `INSERT` detectadas no delta e armazenadas no vetor de operações gerado pelo Algoritmo 2.

O Algoritmo 7 recebe como artefatos de entrada o vetor com as operações correspondentes à inserção de tupla e o documento XML com as chaves geradas, o qual contém as chaves que identificam as tuplas nas relações. Como saída é obtido o conteúdo a ser inserido no banco de dados, que por sua vez é encaminhado juntamente com o identificador da fonte ao componente XMap. Para isso, o laço de repetição percorre o vetor com as operações e se a operação é do tipo `INSERT` (linha `1`), então a posição do nodo modificado é lida no documento com as chaves e o conteúdo correspondente é armazenado na variável `conteúdo`, linha `4`.

Portanto, para cada operação de atualização equivalente à inserção de tupla no banco de dados, o XMap recebe do componente Gerenciador de Atualizações o conteúdo

correspondente à inserção e os mapeia para instruções de inserção na linguagem SQL. Durante o processo de mapeamento, conflitos podem ocorrer, os quais são tratados de maneira a obedecer às restrições de integridade do banco de dados relacional.

Algoritmo 7: Buscar operações de inserção

Entrada: `vetorOperações` {vetor com às operações correspondentes em SQL}
`chavesGeradas` {documento XML com as chaves geradas, versão atualizada}
Saída: `conteudoInserção` {vetor com as informações das operações de inserção}

1. **for all** `operação="INSERT"` em `vetorOperações` **do**
2. `posição` ← `posição` de `operação`
3. `arquivo` ← `chavesGeradas`
4. `conteúdo` ← ler(`posição` em `arquivo`)
5. `fonte` ← atributo `id` de `fonte`
6. Encaminhar `conteudo` e `fonte` para XMap
7. **end for**

O XMap dividiu o processo de montagem da instrução em duas etapas, denominadas de cabeçalho e contêiner. O cabeçalho contém o nome da tabela e as respectivas colunas e o contêiner as tuplas e os respectivos valores. Na Figura 3.17 é ilustrado o esquema de cabeçalho criado para a instrução de *insert* com o conjunto de tuplas correspondentes.

Cabeçalho

```
INSERT INTO <tabela> (<coluna 1>, <coluna 2>, <coluna 3>, ..., <coluna n>) VALUES
```

Tupla(s)

```
(valor_1 coluna 1>, valor_1 coluna 2>, valor_1 coluna 3>, ..., valor_1 coluna n>),
(valor_2 coluna 1>, valor_2 coluna 2>, valor_2 coluna 3>, ..., valor_2 coluna n>),
(valor_3 coluna 1>, valor_3 coluna 2>, valor_3 coluna 3>, ..., valor_3 coluna n>),
...
(valor_m coluna 1>, valor_m coluna 2>, valor_m coluna 3>, ..., valor_m coluna n>),
```

Figura 3.17. Esquema de cabeçalho para a instrução de *insert* em SQL (AVELAR; SACCOL, PIVETA, 2012).

De acordo com a Figura 3.17, uma tabela pode ter até `n` colunas que correspondem aos campos que recebem os dados. O contêiner, que representa as tuplas, pode possuir até `m` tuplas. Para existir compatibilidade da tupla com a instrução de inserção do cabeçalho, a quantidade de valores da tupla deve ser igual à quantidade de campos para inserção de dados.

Para compor o cabeçalho, inicialmente, é obtido o nome da tabela junto ao documento de mapeamento gerado a partir dos documentos XML com as chaves, através de uma busca pelos conceitos não-léxicos. A partir disso, são buscados os conceitos léxicos – que representam colunas da tabela – para compor o restante do cabeçalho seguindo a estrutura da cláusula *insert* de SQL. Para associar as colunas com a tabela é verificado em todos os

conceitos léxicos do documento de mapeamento a presença dos campos que correspondem à tabela em questão. Caso a correspondência seja verdadeira, o campo é adicionado ao cabeçalho para compor as colunas na instrução.

Tendo criado a estrutura da instrução, a etapa seguinte é extrair de cada fonte XML o conteúdo endereçado pelo caminho XPath, presente em cada fonte mapeada no documento de mapeamento. Para isso, a fonte de dados XML é percorrida na busca pelo identificador do arquivo e o caminho XPath. Havendo compatibilidade da expressão XPath do documento de mapeamento com a expressão XPath da fonte, o conteúdo é extraído e adicionado à estrutura de cabeçalho previamente gerado. Conforme exemplificado na Figura 3.18, a expressão de caminho do elemento `<first>` (linha 21) no documento `Doc_A.xml` aponta para a coluna `first` (linha 214 do documento de mapeamento) da tabela `names` (linha 215 do documento de mapeamento). Portanto, o conteúdo do elemento é extraído e adicionado à estrutura da instrução. O mesmo procedimento é repetido para os nodos `@cod_names` (linha 19) e `<last>` (linha 20) do `Doc_A.xml`.

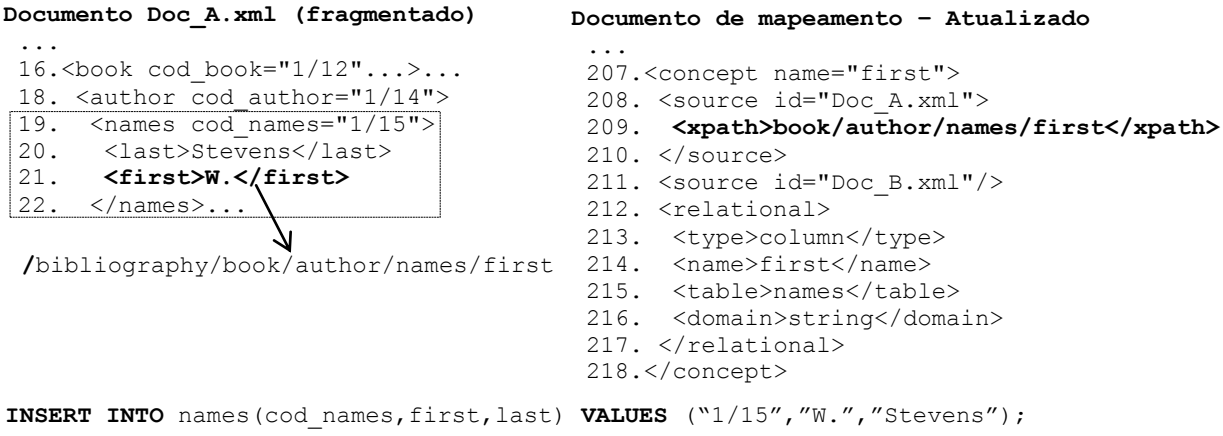


Figura 3.18. Exemplo de composição da instrução de inserção.

Como resultado, tem-se o *script* de inserção de dados em SQL. A partir de então, é possível a execução em banco de dados para armazenar os dados nas tabelas relacionais correspondentes.

3.8. Considerações Finais

O entendimento das etapas de detecção de modificações, geração das chaves XML e propagação das modificações para o banco de dados através do rastreamento das chaves é de

suma importância para o contexto deste trabalho. Para cada etapa são apresentadas funcionalidades, as quais foram incorporadas à arquitetura proposta para a abordagem de propagação de atualizações do *framework* X2Rel.

Uma vez projetada a arquitetura dos componentes propostos, foi detalhado o processo de trabalho dos mesmos. A definição das chaves, realizada pelo componente XKGen (externalizado na arquitetura do X2Rel), é feita com base na equivalência dos conceitos XML no esquema lógico relacional. Logo, o conjunto de dados XML que correspondem aos registros da tupla no BDR recebem a chave gerada a partir do identificador do nodo no documento XML. Com isso, as modificações detectadas nos dados XML fonte podem ser propagadas para os dados relacionais correspondentes através do rastreamento das chaves, o qual é realizado pelo componente DBUpdater.

Operações de modificação detectadas nos documentos XML e informadas no arquivo delta podem ter um significado diferente no BDR. Mais especificamente, o modelo de operações do algoritmo usado para detectar modificações nos dados XML opera sobre uma estrutura de dados diferente do modelo de operações da linguagem SQL, usada para produzir as instruções de atualização sobre o banco de dados. De maneira a estabelecer a equivalência entre as operações nos dois modelos, definiram-se regras de compatibilidade para traduzir a operação de um modelo para o outro.

Entretanto, conflitos podem ocorrer no processo de tradução de operações de modificação em XML para o BDR. Dentre estes, conflitos decorrentes da inserção e deleção de tuplas. Os conflitos de inserção são resolvidos pelo componente XMap, apresentado em (AVELAR; SACCOL, PIVETA, 2012) e incorporado à arquitetura do DBUpdater para tratar a operação de atualização `INSERT`. Já para solucionar os conflitos de deleção, buscou-se identificar o conteúdo deletado e traduzi-lo para tantas instruções de `DELETE` em SQL quanto for necessário.

Algoritmos foram definidos para cada tipo de operação de atualização de maneira a formalizar o processo de tradução das modificações para instruções de atualização sobre o banco de dados, bem como para gerar as chaves e identificar a equivalência entre as operações de modificação nos dados XML e relacional. Os artefatos de saída gerados por outros componentes do X2Rel são necessários para o funcionamento dos algoritmos em questão.

Por último, são geradas as instruções de atualização em linguagem SQL para serem executadas no banco de dados. O cabeçalho das instruções é preenchido de acordo com as informações das operações de modificação detectadas nos documentos XML de origem.

4. EXPERIMENTOS E AVALIAÇÃO DOS RESULTADOS

A abordagem do componente DBUpdater no *framework* X2Rel e as soluções propostas para manutenção do BDR que armazena dados de documentos XML devem ser verificadas, de maneira a validar a abordagem proposta.

Os experimentos correspondem aos testes realizados com documentos XML estruturalmente heterogêneos, representando dados pertencentes a um domínio de conhecimento comum. A partir disso, o processo seguinte passa a ser a análise dos artefatos produzidos pelos componentes OntoGen, OntoRel, CMap e XMap, respectivamente. A detecção das operações de modificação, a geração das chaves, a definição de equivalências entre os modelos de operações, juntamente a solução dos conflitos decorrentes, fazem parte da verificação do componente DBUpdater.

4.1. Arquivos XML

O conjunto de documentos XML que formam a base para os testes neste trabalho deve possuir uma variedade estrutural e de informações para permitir uma execução mais abrangente de experimentos. A internet foi o meio mais apropriado para busca das fontes de dados XML devido a grande variedade de repositórios com domínio público para os dados, os quais podem ser obtidos e compartilhados pelos mais diversos tipos de aplicações.

Direcionado para um conteúdo mais específico, foram selecionadas as fontes de dados do repositório bibliográfico DBLP (*Digital Bibliography & Library Project*) (DBLP, 2013), arbitrariamente escolhidas. O DBLP é constituído de um banco de dados que armazena dados bibliográficos de publicações na área de ciência da computação, idealizado há pelo menos três décadas. Além de possuir um utilitário para navegação em suas fontes de dados, o repositório também disponibiliza representações das informações através de documentos XML, permitindo que os dados publicados possam ser interpretados por outras aplicações.

Essencialmente, os documentos XML são constituídos por informações de publicações bibliográficas, dentre elas, de conferências, revistas e livros. Neste estudo de caso, o domínio de conhecimento adotado está relacionado a informações de publicações em conferências,

dentre as quais foi selecionado a conferência VLDB (*Very Large Data Bases*) e ER (*International Conference on Conceptual Modeling*), ambas com foco na área de gerenciamento de dados.

Em razão de cada publicação ser representada em um arquivo individual, convencionou-se um documento XML que engloba dados de uma coleção de documentos fontes relacionados à conferência. Além disso, alterações foram realizadas na estrutura dos documentos de maneira a fornecer maior heterogeneidade aos mesmos. O conteúdo dos documentos manteve-se inalterado. Com isso, adicionou-se uma maior quantidade de informações aos arquivos de testes, possibilitando experimentos mais abrangentes. A Tabela 4.1 apresenta o endereço das fontes de dados escolhidas em cada conferência e o respectivo nome do arquivo gerado a partir das informações das fontes.

Tabela 4.1. Relação das conferências com as respectivas fontes XML experimentais.

Conferência	**Fonte**	**Documento XML**
VLDB	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/2013bd3.xml	`vldb.xml`
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/BoleyKKSS13.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/2013adms.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/WillhalmO0F13.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/SasakiA13.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/2012adms.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/KimuraGK12.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/vldb/FaustSKP12.xml	
ER	http://dblp.uni-trier.de/rec/bibtex/conf/er/2013.xml	`er.xml`
	http://dblp.uni-trier.de/rec/bibtex/conf/er/EmbleyL13.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/Aufaure13.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/Kaschek13.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/2012.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/CeriVPT12.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/Soussi12.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/2012w.xml	
	http://dblp.uni-trier.de/rec/bibtex/conf/er/BuchmayrKK12.xml	

Partindo da Tabela 4.1, dois documentos XML foram gerados para os ensaios, conforme mostra a coluna `Documento XML`. O documento `vldb.xml` incorpora informações de publicações da conferência VLDB, as quais estão presentes em oito arquivos fonte. Já o documento `er.xml` é composto por nove arquivos relacionados à publicações na conferência ER.

Para facilitar a visualização e entendimento dos arquivos utilizados experimentalmente, foi criado um repositório próprio onde os mesmos podem ser obtidos (SCHUCH, 2014).

4.2. Ontologia Representativa

Reunidas as representações das informações correspondentes a um domínio de conhecimento comum, o componente OntoGen tem a função de gerar a ontologia descritiva das instâncias de dados XML de maneira a abstrair as diferenciações estruturais dos documentos XML para uma única estrutura. A tarefa de determinar o domínio e reunir as correspondentes informações em formato XML para construir a ontologia é delegada ao usuário do *framework* X2Rel.

Considerando o domínio de informações de publicações em conferências, os documentos XML construídos para a base inicial de testes foram utilizados como entrada para OntoGen. A ontologia resultante para os documentos de entrada descreveu cinco conceitos não-léxicos, treze conceitos léxicos do tipo `string` e dois conceitos léxicos do tipo numérico `inteiro`. A representação gráfica da ontologia resultante foi convencionada manualmente a partir da saída gerada por OntoGen, conforme ilustrado na Figura 4.1.

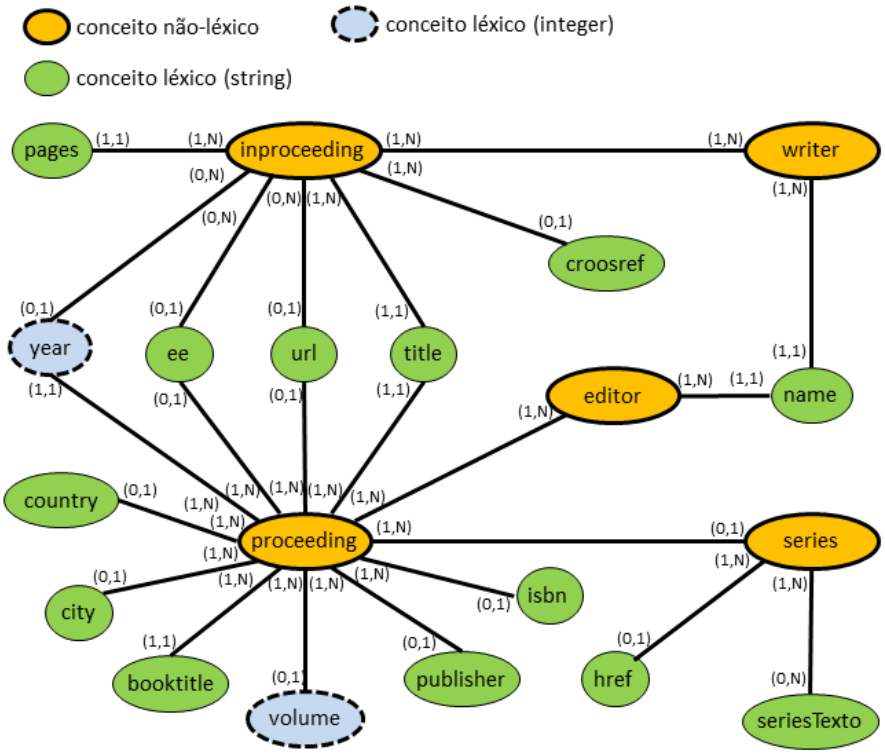


Figura 4.1. Ontologia descritiva dos documentos XML experimentais.

Para melhor compreensão do domínio de conhecimento, a seguir são descritos os conceitos não-léxicos da ontologia ilustrada na Figura 4.1:

- proceeding: corresponde ao registro das informações que descrevem uma conferência ou *workshop*, incluindo informações de título dos anais, país e cidade sede do evento, identificador internacional `isbn`, endereço eletrônico, endereço eletrônico da edição `ee`, volume e ano da publicação;
- inproceeding: representa as informações dos trabalhos publicados na conferência contendo o número das páginas do correspondente trabalho nos anais, título do trabalho, endereço eletrônico, endereço eletrônico da edição `ee`, ano de publicação e chave do registro da conferência `crossref`;
- editor: lista o nome dos principais organizadores da conferência, incluindo o comitê especial e geral. Além disso, o editor dos textos também pode ser listado, que é a pessoa que faz o trabalho de manuscrever as informações disponibilizadas nos anais da conferência;
- writer: lista o nome dos autores e coautores de trabalhos publicados nos anais da conferência;
- series: especifica a série da publicação da conferência, a qual faz parte do volume. Se existir uma série e um volume da publicação, o número do volume é interpretado como a numeração da série. Opcionalmente, `href` contém a URL local da página principal da série.

A modelagem da ontologia representativa do domínio de conhecimento em questão demonstrou como resultados os conceitos equivalentes para as instâncias dos documentos XML. Para isso, o componente OntoGen agrupou as informações de estrutura dos documentos como sinônimos e os representou de maneira padronizada através da ontologia. Além disso, os relacionamentos permitiram definir as dependências entre os conceitos resultantes a partir do conteúdo XML dos documentos experimentais.

4.3. Esquema Lógico Relacional

Gerada a ontologia global representativa dos documentos XML pertencentes ao domínio de conhecimento correspondente, a tarefa de traduzí-la para o esquema lógico relacional é realizada pelo componente OntoRel, detalhado na seção 2.1.1. Considerando que as definições estruturais dos documentos XML são descritas em conceitos não-léxicos,

léxicos e o relacionamento entre os mesmos, o processo de geração do esquema relacional utiliza estas informações para compor sua estrutura.

Com base na ontologia ilustrada na Figura 4.1 para descrever os documentos XML do domínio de publicações bibliográficas, a Figura 4.2 ilustra o esquema lógico relacional resultante da tradução da ontologia. A representação gráfica do esquema relacional da Figura 4.2 foi convencionado a partir do *script* de criação das tabelas gerado por OntoRel. Percebe-se que os conceitos não-léxicos correspondem às tabelas e os conceitos léxicos às colunas da respectiva tabela no esquema do banco de dados. Os tipos de dados determinados na ontologia correspondem ao domínio de dados permitido para as colunas nas relações do banco de dados, como ocorre com `string` na ontologia e `varchar` no esquema relacional. A coluna chave primária da relação é gerada de maneira artificial a partir do nome `cod_` mais o nome da tabela, pois a mesma não está informada na ontologia.

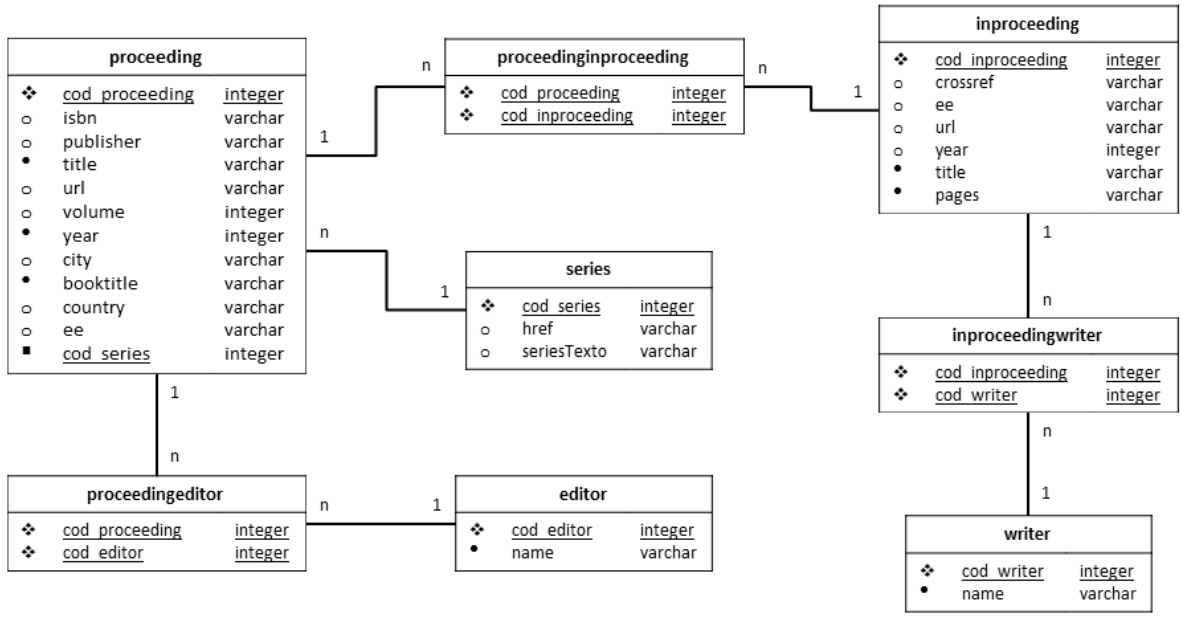


Figura 4.2. Esquema lógico relacional gerado a partir da ontologia experimental.

Dependendo do relacionamento entre conceitos não-léxicos, uma nova relação pode ser gerada, como no caso dos conceitos `proceeding` e `inproceeding`, que resultaram na tabela `proceedinginproceeding`. O mesmo ocorreu para os conceitos `proceeding` e `editor` na ontologia e a tabela `proceedingeditor` no esquema relacional.

4.4. Documento de Mapeamento

As diferenciações estruturais dos documentos XML são abstraídas para uma estrutura única a partir da ontologia representativa do domínio de conhecimento e do esquema lógico relacional consequente. O documento de mapeamento associa os documentos XML com a estrutura da ontologia e do esquema relacional.

Para confeccionar o documento de mapeamento, o CMap associa as informações presentes nos documentos XML da base experimental com os conceitos da ontologia e o esquema relacional. Conforme o fragmento do documento de mapeamento gerado por CMap e ilustrado na Figura 4.3, o atributo `name` (linha 233) do elemento `<concept>` referencia cada conceito da ontologia que tem o conteúdo apontado por um caminho `<xpath>` (linha 235) do arquivo fonte `<source>` (linha 234) identificado pelo atributo `id`.

```
...
233.<concept name="name">
234. <source id="er.xml">
235.  <xpath>/dblp/proceeding/inproceeding/writer/name</xpath>
236. </source>
237. <source id="vldb.xml">
238.  <xpath>/dblp/proceeding/inproceeding/author/name</xpath>
239. </source>
240. <relational>
241.  <type>column</type>
242.  <name>name</name>
243.  <table>writer</table>
244.  <domain>string</domain>
245. </relational>
246.</concept>
```

Figura 4.3. Fragmento do documento de mapeamento experimental.

Além de associar o conteúdo dos documentos XML fonte à cada conceito da ontologia, também são configurados os parâmetros do esquema lógico relacional, indicados pelo elemento `<relational>` (linha 240). Por exemplo, o conteúdo referente ao conceito `name` (linha 233), apontado pelo caminho XPath, será armazenado na coluna `name` (linha 242) da tabela `writer` (linha 243) como uma `string` (linha 244). O documento de mapeamento também é utilizado nas etapas de geração das chaves e resolução de conflitos decorrentes do processo de propagação de atualizações para o banco de dados.

4.5. Definição das Chaves

Assumindo a inexistência do elemento identificador de registro em qualquer fonte de dados XML, deparou-se com a necessidade de um meio que permita estabelecer uma condição para o rastreamento dos dados armazenados no BDR, principalmente no processo de atualização do banco de dados. Contudo, cada tabela do esquema relacional contém uma coluna chave primária criada de maneira artificial por OntoRel, a qual é mapeada como atributo chave no documento XML. O valor da chave é definido a partir do identificador do nodo.

A abordagem do componente DBUpdater proposta neste trabalho definiu um componente específico para gerar as chaves para conteúdo dos documentos XML. Com isso, as operações de modificação detectadas nos dados XML fonte são lidas no documento com as chaves geradas e propagadas para o BDR usando a chave previamente gerada como filtro para identificar os registros. O componente para gerar as chaves, definido nesta dissertação, é externalizado à arquitetura do X2Rel para que também possa ser utilizado na etapa de geração das chaves para dados XML a serem inicialmente armazenados no banco de dados.

4.5.1. Geração e Atribuição dos Identificadores

O processo de geração, atribuição e manutenção do identificador dos nodos do documento XML pode ser verificado observando o fragmento do documento `er.xml` da base de experimentação representado graficamente na Figura 4.4. Devido à quantidade de elementos, reticências (...) são usadas para indicar a existência de *tokens* que foram suprimidos da árvore XML. Os numeradores presentes em cada rótulo apontam as dimensões do *token*, usadas para manter sua posição e ordem na estrutura do documento XML. Os elementos hachurados correspondem à inserção ou remoção de *tokens*. Tal fato ocorre com os elementos `<inproceeding>` (identificado com o rótulo [95,15,2,16]) e seus elementos internos, `<publisher>` (identificado com o rótulo [94,9,2,11]) e `<year>` (identificado com o rótulo [4,3,2,5]), sendo que o último corresponde à operação de remoção.

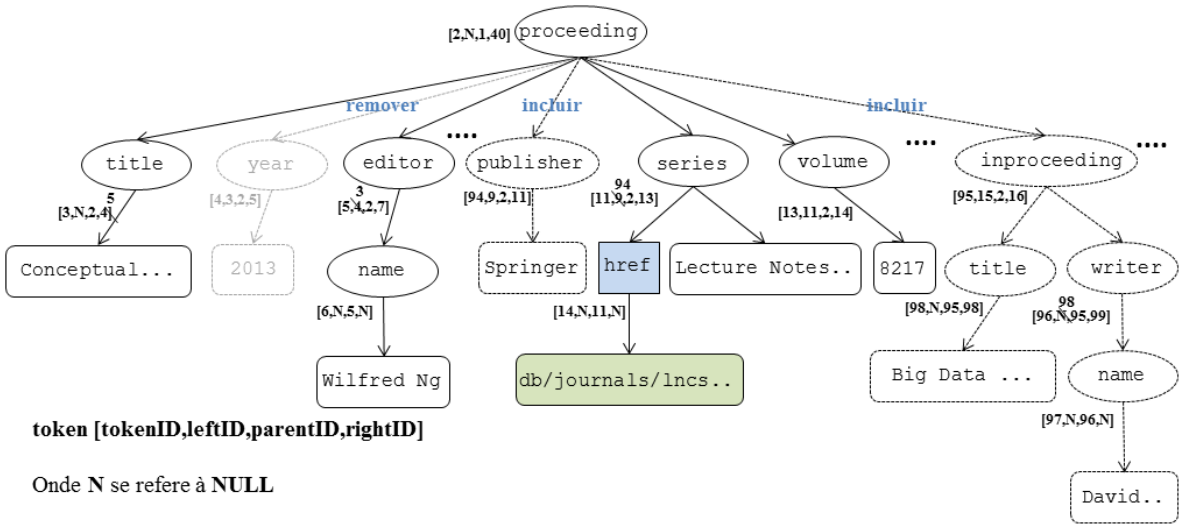


Figura 4.4. Geração, atribuição e manutenção do identificador dos nodos.

Um rótulo identificador é gerado para cada *token*, onde, além do identificador global, são atribuídas três dimensões indicando o *token* irmão à esquerda, o *token* pai e o *token* irmão à direita. Documentos heterogêneos podem incluir em sua estrutura elementos complexos/mistos, como é o caso do elemento `<series>` (identificado com o rótulo [11,94,2,13]), que além de possuir um valor associado tem o atributo `href` (identificado com o rótulo [14,N,11,N]) como descendente.

O processo de manutenção dos rótulos baseia-se no modelo de rotulagem apresentado na seção 2.4.1, onde regras são utilizadas para garantir que o identificador global do *token* mantenha-se constante, sem necessidade de re-rotulagem. Partindo da ilustração da Figura 4.4, pode-se observar duas inserções e uma exclusão de *tokens* na representação do documento `er.xml` da base experimental. Caso a inserção for de subárvore, cada *token* é tratado por uma regra específica para manutenção do rótulo, que é determinada conforme a posição do *token*, como é o caso do elemento `<inproceeding>` e os respectivos elementos internos. Por outro lado, a deleção é tratada como um único *token*.

As regras de manutenção consideradas no exemplo da tabela 4.2 estão presentes no esquema de rotulagem utilizado nesta abordagem. Em razão da quantidade de regras, convencionou-se listar apenas as regras mais gerais para ilustrar os casos de inserção e deleção, seguindo a ordem, o tipo e a posição em que a operação é executada na árvore XML exemplificada na Figura 4.4. As regras estão detalhas na seção 2.4.1.

Tabela 4.2. Exemplos de regras para manutenção de rótulos identificadores.

Ordem	Rótulo	Tipo	Posição	Regra
1	[95,15,2,16]	inserção	à direita	B
2	[96,98,95,99]	inserção	como filho	C
3	[97,N,96,N	inserção	como filho	C
4	[98,N,95,98]	inserção	à esquerda	A
5	[94,9,2,11]	inserção	entre irmãos	D
6	[4,3,2,5]	deleção	entre irmãos	A

Para verificar a eficácia do processo de manutenção dos rótulos, a seguir são aplicadas as regras referentes à ordem 1, 2, 4, 5 e 6, de maneira a validar sua aplicabilidade nos documentos experimentais. As regras são aplicadas com base na ilustração gráfica do documento experimental `er.xml` da Figura 4.4.

- Ordem 1: Inserção do elemento `<inproceeding>` à direita do elemento candidato `<booktitle>`, identificado com o rótulo [15,14,2,95], porém, suprimido na ilustração da Figura 4.4.

  1) `tokenID(inproceeding)` ← `maxTokenID (94) + 1`
  2) `leftID(inproceeding)` ← `tokenID(booktitle)`
  3) `rightID(inproceeding)` ← `Null`
  4) `rightID(booktitle)` ← `tokenID(inproceeding)`
  5) `parentID(inproceeding)` ← `parentID(booktitle)`

- Ordem 2: Inserção do elemento `<writer>` como filho do elemento candidato `<inproceeding>`.

  1) `tokenID(writer)` ← `maxTokenID (95) + 1`
  2) `leftID(writer)` ← `Null`
  3) `rightID(writer)` ← `Null`
  4) `parentID(writer)` ← `tokenID(inproceeding)`

- Ordem 4: Inserção do elemento `<title>` à esquerda do elemento candidato `<writer>`.

  1) `tokenID(title)` ← `maxTokenID (97) + 1`
  2) `rightID(title)` ← `tokenID(writer)`
  3) `leftID(title)` ← `Null`
  4) `leftID(writer)` ← `tokenID(title)`
  5) `parentID(title)` ← `parentID(writer)`

- Ordem 5: Inserção do elemento `<publisher>` entre os elementos irmãos `<editor>` (suprimido da ilustração), identificado com o rótulo [9,7,2,94], e `<series>`.

1) tokenID(publisher) ← maxTokenID (93) + 1
2) rightID(publisher) ← rightID(editor)
3) leftID(publisher) ← leftID(series)
4) rightID(editor) ← tokenID(publisher)
5) leftID(series) ← tokenID(publisher)
6) parentID(publisher) ← parentID(series)

- Ordem 6: Deleção do elemento <year> localizado entre os elementos irmãos <title> e <editor>.

  1) rightID(title) ← rightID(year)
  2) leftID(editor) ← leftID(year)

Para os casos de inserção, a regra 1 tem a função de gerar o valor do identificador global seguido do identificador do último *token*. As demais regras possuem a função de manter o identificador global como constante, atualizando apenas as relações pai-filho, ancestral-descendente e de irmãos. As informações do rótulo identificador juntamente com as propriedades do *token* são de interesse no processo de geração da chave identificadora das tuplas, sendo materializadas em um documento de mapeamento (ilustrado na Figura 4.5) de maneira que possam ser utilizadas na etapa de geração das chaves para o documento fonte.

```
 1. <mapping>
 2.  <document id="2">
 3.   <source name="er.xml"/> ...
13.   <token id="2" left="" parent="1" right="30">
14.    <source name="proceeding">
15.     <xpath>/dblp/proceeding</xpath>
16.    </source>
17.    <properties>
18.     <type>element</type>
19.    </properties>
20.   </token>
21.   <token id="3" left="" parent="2" right="5">
22.    <source name="title">
23.     <xpath>/dblp/proceeding/title</xpath>
24.    </source>
25.    <properties>
26.     <value>Conceptual...</value>
27.     <type>element</type>
28.    </properties>
29.   </token>...
```

Figura 4.5. Fragmento do documento de mapeamento dos *tokens* do documento experimental er.xml.

O documento de mapeamento dos *tokens* resultante informa a localização e as propriedades dos elementos e atributos presentes no arquivo de publicações bibliográficas em formato XML. Conforme o fragmento ilustrado na Figura 4.5 retirado do documento de

mapeamento dos *tokens*, o documento XML está identificado unicamente pelo atributo `id` do elemento `<document>` (linha 2), bem como a identificação de cada um de seus *tokens* pelo atributo `id` do elemento `<token>` que, juntamente com os atributos `left`, `parent` e `right` (linha 13), formam o rótulo identificador da posição do *token*. O nome do *token*, designado pelo atributo `name` (linha 14), é apontado por um caminho em `<xpath>` (linha 15) e suas propriedades estão delimitadas no elemento `<properties>` (linha 17). O documento de mapeamento dos *tokens* dos arquivos experimentais pode ser acessado integralmente em repositório próprio (SCHUCH, 2014).

4.5.2. Geração das Chaves

O esquema lógico relacional, criado pelo componente OntoRel a partir da ontologia, constitui a estrutura das tabelas relacionais que contém o atributo chave primária criado de maneira artificial. Tal atributo é designado como atributo chave no documento XML, onde recebe como valor o identificador do documento concatenado com o identificador do *token*, tendo a função de identificar unicamente cada registro no BDR. Como exemplo, a Figura 4.6 exibe o fragmento do documento XML com as chaves geradas para o documento `er.xml`, representado graficamente na Figura 4.4.

```
 ...
 2. <proceeding cod_proceeding="2/2">
 3.  <title>Conceptual...</title>
 4.  <year>2013</year>
 5.  <editor cod_editor="2/5">
 6.   <name>Wilfred Ng</name>
 7.  </editor>
 8.  <editor cod_editor="2/7">
 9.   <name>Veda C. Storey</name>
10.  </editor>...
14.  <publisher>Springer</publisher>
15.  <series cod_series="2/11" href="db...">Lecture...</series>
16.  <volume>8217</volume>
17.  <isbn>978-3-642-41923-2</isbn>
18.  <booktitle>ER</booktitle>
19.  <inproceeding cod_inproceeding="2/95">
20.   <title>Big Data...</title>
21.   <writer cod_writer="2/97">
22.    <name>David W. Embley</name>
23.   </writer>...
131.</proceeding>
```

Figura 4.6. Documento `er.xml` com as chaves geradas.

Para todo conjunto de dados XML correspondente à tupla, o documento XML recebe o atributo chave primária que permite rastrear os dados XML com a tupla relacional. Durante o processo de mapeamento do atributo chave, o documento de mapeamento é utilizado para determinar o *token* que recebe o identificador de registro. O documento exemplificado tem como identificador o valor `2` (linha 2 do documento de mapeamento dos *tokens* da Figura 4.5), o qual é concatenado com o identificador do *token* equivalente a tabela, separado pelo caractere "`/`", formando a chave identificadora.

Para um melhor entendimento, pode-se analisar a chave `@cod_editor=2/5` mostrada na linha 5 da Figura 4.6. Neste caso, o elemento `<editor>` (linha 5) é conceitualmente equivalente a tabela `editor` no esquema relacional, e portanto recebeu o atributo chave primária `cod_editor` da correspondente tabela. Tal atributo tem a função de chave identificadora da subárvore `<editor>`, a qual corresponde a uma tupla na tabela editor. Como valor, a chave recebe o identificador do documento de origem (`er.xml`), que é `2`, concatenado com o identificador do elemento `<editor>`, que é `5`. O caractere separador é utilizado para evitar a formação de chaves ambíguas.

4.5.3. Atualização da Ontologia Descritiva e Documento de Mapeamento

Geradas as chaves para os documentos XML fonte, a etapa seguinte é atualizar a correspondente ontologia de maneira que os conceitos dos atributos chave também sejam descritos. Do mesmo modo, o documento de mapeamento dos conceitos também é atualizado para representar a equivalência do conceito dos atributos chave, a ontologia e o esquema relacional. O processo de atualização é realizado pelo usuário do *framework* X2Rel, de maneira a reconstruir os artefatos anteriormente citados.

A representação gráfica da Figura 4.7 mostra o resultado da atualização da ontologia representativa ilustrada na Figura 4.1, descrevendo os conceitos dos atributos chave gerados para os documentos XML de experimentação. De acordo com a ilustração, foram incluídos cinco conceitos léxicos do tipo numérico `inteiro`, sendo um para cada conceito não-léxico.

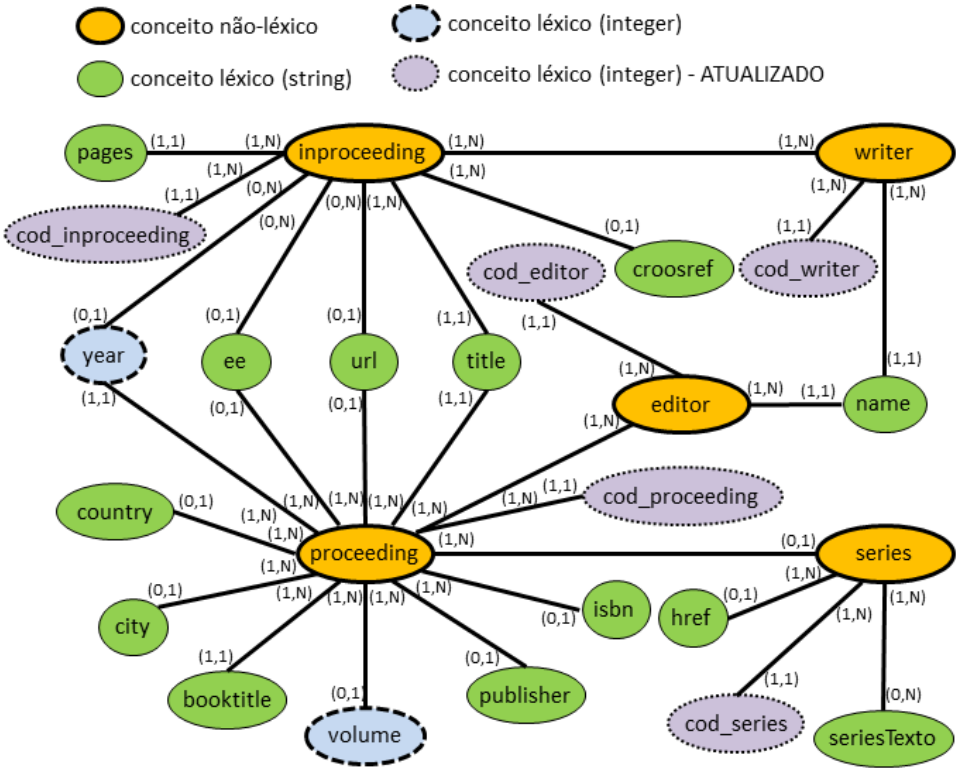


Figura 4.7. Ontologia descritiva atualizada a partir das chaves geradas.

Assim como na ontologia, o documento de mapeamento também foi atualizado de maneira a incluir a representação da equivalência para os conceitos gerados a partir dos atributos chave. Desse modo, as propriedades de cinco novos conceitos léxicos foram mapeadas ao documento de mapeamento, são eles: `cod_inproceeding`, `cod_editor`, `cod_writer`, `cod_proceeding` e `cod_series`. O fragmento do documento de mapeamento da Figura 4.8 mostra as informações referentes ao conceito `cod_inproceeding` (linha 325).

```
...
325.<concept name="cod_inproceeding">
326. <source id="er.xml">
327.  <xpath>/dblp/proceeding/inproceeding/cod_inproceeding</xpath>
328. </source>
329. <source id="vldb.xml">
330.  <xpath>/dblp/proceeding/inproceeding/cod_inproceeding</xpath>
331. </source>
332. <relational>
333.  <type>column</type>
334.  <name>cod_inproceeding</name>
335.  <table>inproceeding</table>
336.  <domain>integer</domain>
337. </relational>
338.</concept>
```

Figura 4.8. Fragmento do documento de mapeamento atualizado.

Conforme exemplificado na Figura 4.8, o conteúdo referente ao conceito `cod_inproceeding` (linha 325), aponta, através do caminho XPath (linha 327), para a coluna `cod_inproceeding` (linha 333) da tabela `inproceeding` (linha 335) como valor numérico (linha 336). O documento de mapeamento atualizado pode ser acessado de maneira integral em (SCHUCH, 2014).

4.6. Inserção dos dados

O esquema lógico relacional, gerado pelo componente OntoRel a partir da ontologia, constitui a estrutura do BDR. A partir da estrutura estabelecida, a próxima etapa é a inserção dos dados a partir dos documentos XML com as chaves geradas para o BDR através do componente XMap do *framework* X2Rel.

Conforme apresentado na Tabela 4.1, são dois documentos convencionados para a realização dos testes. Como exemplo, a Figura 4.9 exibe a instrução SQL de inserção do conteúdo XML na tabela `inproceeding` do banco de dados. Os valores da chave primária `cod_inproceeding` foram mapeados a partir das chaves geradas.

`vldb.xml` (original)

INSERT INTO inproceeding(cod_inproceeding,crossref,ee,url,year,title,pages) VALUES
(“1/19”,null,null,null,null,”Communication-efficient Outlier Detection for Scale-out Systems.”,”19-24”),
(“1/42”,null,null,null,null,”Vectorizing Database Column Scans with Complex Predicates.”,”1-12”),
(“1/53”,null,null,null,2013,”Modularizing B+-trees: Three-Level B+-trees Work Fine.”,”46-57”),
(“1/71”,null,null,null,null,”Efficient Locking Techniques for Databases on Modern Hardware.”,”1/12”),
(“1/80”,null,null,null,null,”Fast Lookups for In-Memory Column Stores: Group-Key Indices, Lookup and Maintenance.”,”13-22”);

`er.xml` (original)

INSERT INTO inproceeding(cod_inproceeding,crossref,ee,url,year,title,pages) VALUES
(“2/16”,”conf/er/2013”,null,”http://dx.doi.org/10.1007/978-3-642-41924-9_2”,”db/conf/er/er2013.html#Aufaure13”,”What's Up in Business Intelligence? A Contextual and Knowledge-Based Perspective.”,”9-18”),
(“2/24”,null,null,” db/conf/er/er2013.html#Kaschek13”,null,” A Semantic Analysis of Shared References.”,”88-95”),
(“2/47”,null,null,”db/conf/er/er2012.html#CeriVPT12”,” Mega-modeling for Big Data Analytics.”,”1-15”),
(“2/59”,null,null,”db/conf/er/er2012.html#Soussi12”,null,” SPIDER-Graph: A Model for Heterogeneous Graphs Extracted from a Relational Database.”,”543-552”),
(“2/84”,null,null,” db/conf/er/erw2012.html#BuchmayrKK12”,null,”A Rule Based Approach for Mapping Sensor Data to Ontological Models in AAL Environments”,”3-12”);

Figura 4.9. Inserção de conteúdo na tabela `inproceeding` do banco de dados.

Para facilitar a visualização do resultado das instruções a Figura 4.10 mostra a tabela `inprocceding` com o conteúdo dos documentos `vldb.xml` e `er.xml` armazenados.

inproceeding

cod_inproceeding	crossref	ee	url	year	title	pages
1/19					Communication-efficient Outlier Detection for Scale-out Systems.	19-24
1/42					Vectorizing Database Column Scans with Complex Predicates.	1-12
1/53				2013	Modularizing B+-trees: Three-Level B+-trees Work Fine.	46-57
1/71					Efficient Locking Techniques for Databases on Modern Hardware.	1-12
1/80					Fast Lookups for In-Memory Column Stores: Group-Key Indices, Lookup and	13-22
2/16	conf/er/2013	http://dx.doi.org/10.1007/978-3-642-41924-9_2	db/conf/er/er2013.html#Aufaure13		What's Up in Business Intelligence? A Contextual and Knowledge-Based	9-18
2/24			db/conf/er/er2013.html#Kaschek13		A Semantic Analysis of Shared References.	88-95
2/47			db/conf/er/er2012.html#CeriVPT12		Mega-modeling for Big Data Analytics.	1-15
2/59			db/conf/er/er2012.html#Soussi12		SPIDER-Graph: A Model for Heterogeneous Graphs Extracted from a Relational	543-552
2/84			db/conf/er/erw2012.html#BuchmayrKK12		A Rule Based Approach for Mapping Sensor Data to Ontological Models in AAL	3-12

Figura 4.10. Tabela `inproceeding` com o conteúdo dos documentos XML armazenados.

Os documentos de experimentação constituem 10 linhas de inserção de dados na tabela `inproceeding`, conforme resultado mostrado na Figura 4.10. Destas 10 linhas, 5 são provenientes do documento `vldb.xml` e 5 do `er.xml`, como referenciado na Figura 4.9. Há linhas cujo valor dos campos são iguais a `nulo`, sendo que estes são tratados por XMap como conflito de elemento inexistente, que ocorre quando não há informação de um elemento para ser armazenado na tabela. Os campos `nulo` são mantidos na instrução de maneira a corresponder com a quantidade de campos da tabela.

Como resultado do processo de mapeamento do conteúdo XML em BDR tem-se o *script* de inserção de dados em SQL. A partir disso, é possível a execução em banco de dados para armazenar os dados no esquema relacional previamente gerado. O *script* de inserção completo pode ser obtido em (SCHUCH, 2014).

4.7. Detecção do Delta

A partir da versão original e modificada de cada documento XML de experimentação, o componente Detector de Deltas, através do algoritmo X-Diff, tem a função de detectar as

diferenças entre as versões e informá-las através de operações de edição em um arquivo no formato XML, chamado de delta. Tais operações são usadas como base, para enfim, atualizar a base relacional.

Em razão de não ter sido encontrado um repositório com versões dos documentos XML já formadas, convencionou-se as versões para os documentos de experimentação de maneira manual, ou seja, as modificações nos dados fonte foram feitas especificamente para a realização dos testes. Para exemplificar, a Figura 4.11 ilustra dois arquivos delta gerados a partir das versões dos documentos de experimentação, onde um é relativo ao documento `vldb.xml` e outro ao documento `er.xml`. As versões dos documentos, bem como os arquivos delta, podem ser obtidos integralmente em (SCHUCH, 2014).

vldb.xml (fragmentado)

```
 ...
 2.<proceeding>...
15. <title>Proceedings...</title>...
21. <volume>1020<?UPDATE FROM "1018"?>
    </volume>
24. <inproceeding>
    <?DELETE inproceeding?>
25.  <author>
26.   <name>Moshe Gabel</name>
27.  </author>
28.  <author>
29.   <name>Daniel Keren</name>
30.  </author>
31.  <author>
32.   <name>Assaf Schuster</name>
33.  </author>
34.  <title>Communication...</title>...
36. </inproceeding>...
37.</proceeding>
```

er.xml (fragmentado)

```
...
2.<proceeding>
3. <title>Conceptual...</title>
4. <year><?DELETE year?>2013</year>
5. <editor>
6.  <name>Wilfred Ng</name>
7. </editor>...
14. <publisher><?INSERT publisher?>Springer
    </publisher>
15. <series href="db...">Lecture...
    </series>
16. <volume>8217</volume>...
19. <inproceeding><?INSERT inproceeding?>
20.  <title>Big Data...</title>
21.  <writer>
22.   <name>David W. Embley</name>
23.  </writer>...
30. </inproceeding>...
49.</proceeding>
```

Figura 4.11. Deltas gerados pelo algoritmo X-Diff informando as operações de modificação detectadas entre as versões dos documentos de experimentação.

Observando a representação dos arquivos delta da Figura 4.11, é possível identificar duas operações de modificação no arquivo `vldb.xml` e três no arquivo `er.xml`, sendo que as mesmas estão informadas em nodos do tipo instrução de processamento `<? ?>`:

- O delta do documento `vldb.xml` diz que o volume de publicação da conferência `"Proceedings..."` foi modificado para `"1020"` (linha 21). Do mesmo modo, o delta diz que foi deletado da conferência o trabalho intitulado de `"Communication..."`, o qual é de autoria de `"Moshe Gabel"`, `"Daniel Keren"` e `"Assaf Schuster"`, conforme linha 24.

- O delta do documento `er.xml` diz que o ano de realização e publicação foi removido da conferência (linha 4), enquanto o local de publicação foi incluído (linha 14). Além disso, de acordo com à operação informada na linha 19, o delta diz que o trabalho intitulado de "`Big Data...`", cujo um dos autores é "`David W. Embley`", foi adicionado à conferência.

Uma vez que os arquivos delta foram gerados, a etapa seguinte tem a tarefa de identificar e gerar as equivalências entre as operações detectadas no delta em relação ao modelo de operações de atualização da linguagem SQL.

4.8. Equivalência entre Operações

O delta gerado pelo algoritmo X-Diff informa as operações de modificação detectadas entre as versões do documento XML fonte. Assim como na linguagem de atualização SQL, o algoritmo de *Diff* também possui um modelo de operações. De maneira a estabelecer a equivalência entre as operações de um modelo e outro, foram definidas regras de equivalência, conforme apresentado na Tabela 3.1.

Para cada arquivo delta, são buscadas as operações de edição e determinada a equivalência de cada uma em relação à instrução de atualização em linguagem SQL. De maneira geral, operações detectadas sobre nodos folha do documento XML correspondem à atualização de conteúdo do campo da tupla relacional, enquanto inserções ou deleções de subárvores equivalem à inserção ou deleção de tupla.

Partindo dos arquivos delta de experimentação ilustrados na Figura 4.11, a Tabela 4.3 apresenta a equivalência entre as operações.

Tabela 4.3. Exemplo de equivalência para às operações informadas no delta.

Arquivo	**Operação de edição XML**	**Operação equivalente em SQL**
`vldb.xml`	`UPDATE FROM "1018"`	`UPDATE`
`vldb.xml`	`DELETE inproceeding`	`DELETE`
`er.xml`	`DELETE year`	`UPDATE`
`er.xml`	`INSERT publisher`	`UPDATE`
`er.xml`	`INSERT inproceeding`	`INSERT`

Nota-se que o arquivo `vldb.xml` contém duas operações, sendo uma correspondente à atualização e outra à deleção de tupla. Do mesmo modo, o arquivo `er.xml` informa duas operações equivalentes à atualização e uma à inserção de tupla. A equivalência de operações para ambos os arquivos delta são geradas pelo Algoritmo 2, detalhado na seção 3.5.4.

4.9. Solução dos Conflitos Existentes

Identificada a equivalência entre as operações de edição, a etapa seguinte diz respeito à propagação das modificações para o BDR. No entanto, conflitos podem ocorrer durante o processo de tradução de operações detectadas no delta para instruções de atualização correspondentes em SQL. Conforme apresentado na seção 3.6, podem ocorrer conflitos tanto de inserção quanto de deleção, os quais são tratados de maneira a produzir instruções de atualização coerentes com o conteúdo modificado nos documentos XML fonte.

4.9.1. Conflitos de Inserção

Partindo dos arquivos delta exemplificados na Figura 4.11, há pelo menos um conflito a ser tratado na operação correspondente à inserção detectada no arquivo `er.xml`. Tal conflito ocorre em razão da representação do conteúdo de forma distinta do formato exigido para armazenamento em banco de dados. Mais especificamente, o conteúdo presente no documento XML não é compatível com o tipo de dado correspondente no esquema relacional (AVELAR; SACCOL, PIVETA, 2012).

Um exemplo de conflito de representação pode ser observado comparando o tipo de dado dos atributos chave em relação ao tipo de dado da coluna chave primária da tabela relacional. Observando o fragmento do documento `er.xml` da Figura 4.6, o valor do atributo chave `cod_inproceeding` (linha 19) não é compatível com o tipo de dado numérico da coluna relacional `cod_inproceeding` representada no documento de mapeamento da Figura 4.8, conforme a linha 336. Para solucionar este conflito, o tipo de dado é alterado para

`string` em todos os campos chave primária das tabelas do banco de dados, como pode ser observado na ilustração da Figura 4.12.

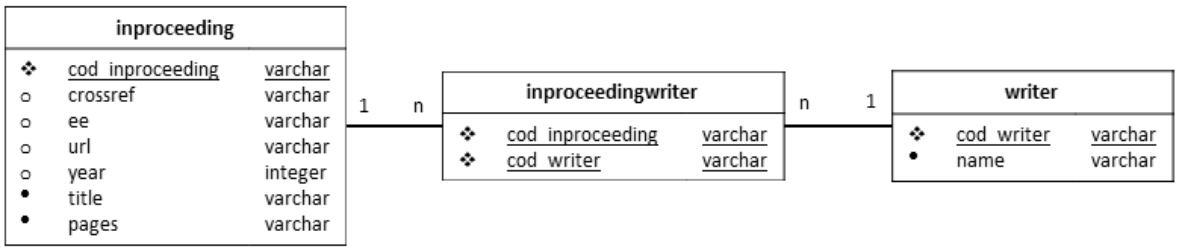


Figura 4.12. Esquema lógico relacional com o tipo de dado alterado.

De acordo com o exemplo ilustrado na Figura 4.12, as colunas `cod_inproceeding` da tabela `inproceeding` e `cod_writer` da tabela `writer` tiveram o domínio de seus dados alterados para o tipo `string` (`varchar`), como mostra a Figura 4.12. Tal alteração foi realizada de maneira convencional a partir do esquema lógico relacional inicialmente gerado por OntoRel, o qual pode ser observado na ilustração da Figura 4.2.

4.9.2. Conflitos de Deleção

Conflitos de deleção ocorrem diante da remoção de subárvores que contém outras subárvores descendentes, o que reflete na deleção de mais de uma tupla do banco de dados. Este conflito é tratado de forma que a operação detectada no delta seja traduzida para tantas instruções de deleção quanto necessário. A posição do elemento que informa a operação de deleção é lida no documento XML com as chaves geradas para identificar a chave de cada instrução de *delete* correspondente.

Observando o arquivo delta `vldb.xml` da Figura 4.11, é possível constatar que a remoção do elemento `<inproceeding>` (linha 24) ocasiona a deleção dos três elementos descendentes `<author>` (linhas 25, 28 e 31), sendo que cada um corresponde a uma subárvore. Portanto, a operação é traduzida para quatro instruções de deleção em linguagem SQL, que de acordo com o documento de mapeamento, refletem nas tabelas `inproceeding` e `writer`.

A Tabela 4.4 exemplifica as instruções de deleção resultantes da operação de detectada no arquivo delta `vldb.xml` da Figura 4.11.

Tabela 4.4. Exemplo de ocorrência do conflito de deleção de múltiplas tuplas.

Tupla	Tabela alvo	Condição da instrução
1	inproceeding	cod_inproceeding=”1/2”
2	writer	cod_writer=”1/20”
3	writer	cod_writer=”1/22”
4	writer	cod_writer=”1/24”

Conforme exemplificado na Tabela 4.4, a operação de deleção detectada sobre o elemento <inproceeding> (linha 24 da Figura 4.11) resultou na remoção de quatro tuplas do banco de dados, tendo como alvos a tabela `inproceeding` (tupla `1`) e `writer` (tupla `2`, `3` e `4`). Para localizar a tupla na tabela relacional, a cada operação de remoção é lida sua posição no documento XML com as chaves geradas para obter a condição da instrução, que é determinada pela chave primária. De acordo com o exemplo da Tabela 4.4, a instrução de deleção da tupla `1` (Tabela 4.4) tem a tabela `inproceeding` como alvo e a chave `cod_inproceeding=”1/2”` como condição. Da mesma forma, as tuplas `2`, `3` e `4` tem como alvo a tabela `writer` e as respectivas chaves, as quais permitem identificar precisamente a tupla no banco de dados.

4.10. Atualização do Banco de Dados

Inseridos os dados nas tabelas do BDR, a próxima etapa constitui o processo de atualização do banco de dados à medida que o conteúdo dos documentos XML sofre alterações. Tal processo é compreendido como a manutenção da visão relacional materializada a partir dos documentos XML.

As operações informadas nos arquivos delta representam a base para o processo de propagação das modificações para a base relacional. Para toda a operação de edição XML é buscada a equivalência no modelo relacional e solucionados possíveis conflitos decorrentes do processo de tradução de XML para relacional. Como exemplo, a Figura 4.13 exibe as

instruções SQL de atualização do conteúdo XML geradas a partir dos arquivos delta `vldb.xml` e `er.xml` da Figura 4.11.

Delta vldb.xml
1.UPDATE FROM proceeding SET volume=”1020” WHERE cod_proceeding=”1/2”;
2.DELETE FROM inproceeding WHERE cod_inproceeding=”1/19";
3.DELETE FROM writer WHERE cod_writer=”1/20”;
4.DELETE FROM writer WHERE cod_writer=”1/22”;
5.DELETE FROM writer WHERE cod_writer=”1/24”;

Delta er.xml
1.UPDATE FROM proceeding SET year=” ” WHERE cod_proceeding=”2/2”;
2.UPDATE FROM proceeding SET publisher=”Springer” WHERE cod_proceeding=”2/2”;
3.INSERT INTO inproceeding(“2/95”,”conf/er/2013”,null,”db/conf/er/er2013.html#EmbleyL13”,null,”Big Data - Conceptual Modeling to the Rescue”,”1-8”);
4.INSERT INTO writer(“2/96”,”David W. Embley”);
5.INSERT INTO writer(“2/99”,”Stephen W. Liddle”);

Figura 4.13. Instruções de atualização do banco de dados resultantes dos arquivos delta.

Note que a condição de cada instrução é determinada pela chave previamente gerada, assim como a correspondente tabela e dados alvo obtidos através do documento de mapeamento gerado pelo componente CMap. Dentre as instruções resultantes mostradas na Figura 4.13, as instruções `2`, `3`, `4` e `5` foram compostas a partir da operação de modificação informada na linha 24 do delta `vldb.xml` da Figura 4.11. Tal resultado foi obtido a partir da tratativa do conflito de deleção de múltiplas tuplas, descrito na seção 4.9.2. Destaca-se ainda, a instrução resultante da operação de deleção informada na linha 4 do arquivo delta `er.xml`. Tal operação foi traduzida para uma instrução de atualização do campo `year` da tabela `proceeding`, cuja tupla identificada pela chave `cod_proceeding=”2/2”` (instrução `1` do delta `er.xml` da Figura 4.13). Nesse caso, o campo da tupla assume valor `nulo`.

A partir das instruções de atualização, é possível a execução em banco de dados para atualizar os dados previamente armazenados.

4.11. Considerações Finais

Para a etapa de realização do estudo de caso e avaliação dos resultados, documentos XML foram convencionados a partir de informações pertencentes a um domínio de conhecimento comum. Em relação ao domínio, o usuário tem a tarefa de identificar os documentos XML cujo domínio de aplicação seja de interesse.

Partindo dos documentos XML formados para o processo de experimentação, as etapas de padronização das estruturas heterogêneas e geração do esquema relacional foram realizadas. Através do componente OntoGen, a ontologia descritiva foi gerada. A partir do componente OntoRel, a ontologia foi traduzida para um esquema lógico relacional único. Por conseguinte, foi definido o documento de mapeamento das equivalências entre os documentos XML, a ontologia e o esquema relacional.

Assumindo a inexistência do elemento identificador de tupla nos documentos XML, a abordagem do componente DBUpdater passou a atuar no processo de definição das chaves para rastrear os dados XML com os relacionais. A partir das chaves geradas, a ontologia descritiva e o correspondente documento de mapeamento foram atualizados para incluir a descrição dos atributos chave.

Para identificar as modificações realizadas nos dados XML, duas versões foram mantidas para cada documento. A partir das versões, um algoritmo de *Diff* foi utilizado para detectar o delta com as operações de modificações. As operações informadas no arquivo delta foram utilizadas como base, para enfim, atualizar a base de dados relacional.

Em razão das operações contidas no arquivo delta, no modelo XML, terem a possibilidade de apresentar comportamento distinto no modelo de atualização da linguagem SQL, foram definidas equivalências para as operações entre os dois modelos. A partir das operações processadas, todas as possíveis equivalências foram testadas no processo de geração da equivalência.

Conflitos de atualização foram analisados para as instruções correspondentes à inserção e deleção. A resolução para conflitos de inserção foi realizada com base na abordagem do componente XMap do *framework* X2Rel, apresentado em (AVELAR; SACCOL, PIVETA, 2012) e incorporado à arquitetura do DBUpdater. Já para resolver o conflito de deleção de múltiplas tuplas foram identificadas todas as tuplas contidas nas operações correspondentes a instrução de *delete* e gerado uma instrução de deleção em linguagem SQL para cada tupla.

Para a análise experimental do componente DBUpdater do *framework* X2Rel, foram utilizados documentos XML de publicações em conferências arbitrariamente selecionadas. A busca foi realizada no repositório de dados bibliográficos DBLP, que possui domínio de divulgação público para as informações.

5. CONCLUSÕES

Visto que o *framework* X2Rel apresenta diferentes funcionalidades que permitem armazenar e consultar dados XML em BDR, o foco deste trabalho foi apresentar a abordagem do DBUpdater, cuja funcionalidade está incorporada ao contexto deste *framework*. O DBUpdater é o componente responsável por detectar e propagar modificações em dados XML para instruções de atualização em BDR.

O diferencial da proposta do DBUpdater está na utilização de trabalhos já existentes nas áreas de detecção de diferenças em dados XML e de atribuição de identificadores aos nodos do documento XML. Unindo abordagens nestas duas áreas pode-se definir uma estratégia que detecta alterações em dados XML e os rastreia no BDR, onde estes dados estão armazenados. No entanto, o processo de propagação das modificações para o banco de dados é constituído por etapas intermediárias de tradução de operações de edição XML para instruções de atualização em SQL.

Decompondo a proposta da abordagem em tarefas, a primeira tem a função de detectar o delta que informa as operações de modificação realizadas nos dados XML. Para isso, adotou-se a estratégia de manter uma versão original e modificada do documento XML e empregar um algoritmo de *Diff* já existente para gerar o delta com as operações de edição. Para definir o algoritmo adequado, este trabalho fez um estudo sobre as principais técnicas de detecção de *Diff* disponíveis na literatura, focando principalmente nos aspectos relevantes para o domínio de aplicação do DBUpdater. Desse modo, o algoritmo X-Diff (WANG; DEWITT; CAI, 2003) foi escolhido principalmente pelo aspecto do seu modelo de árvore e operações, o que o torna adequado em um ambiente que armazena dados XML em BDR.

A principal contribuição deste trabalho está na geração e atribuição de chaves identificadoras para o conteúdo dos documentos XML. Além das chaves geradas serem fundamentais no processo de rastreamento dos dados XML armazenados no banco de dados, o mapeamento e inserção dos dados iniciais nas tabelas relacionais também se beneficia com tal tarefa. De forma diferente das abordagens encontradas na literatura, as chaves são definidas inicialmente nos dados XML e compartilhadas com as tabelas relacionais e não exportadas a partir da chave primária das tabelas para visões XML. Nesta abordagem, as tuplas no documento XML são identificadas com uma chave que permite referenciá-las com as tuplas relacionais pela chave primária das tabelas.

Algumas características tiveram que ser trabalhadas diante do formato de saída do arquivo delta. A principal delas foi estabelecer a equivalência para operações detectadas pelo algoritmo de *Diff* em relação às instruções de atualização sobre o banco de dados, na linguagem SQL DML. Além disso, conflitos precisaram ser resolvidos no processo de tradução de operações entre o modelo XML e relacional. Os algoritmos desenvolvidos para identificar a equivalência de operações e resolução de conflitos consideram como fundamentais os artefatos gerados pelos demais componentes do *framework* X2Rel para propor a abordagem de transformação entre diferentes modelos de dados.

Por fim, a responsabilidade do componente DBUpdater é identificar e propagar modificações de dados XML para operações de atualização em banco de dados através do rastreamento de dados com as chaves geradas. Nesse intermédio, inconsistências existentes devem ser tratadas. Uma vez identificadas as operações de atualização correspondentes, o *script* de edição em banco de dados é criado conforme o esquema lógico relacional de maneira a obedecer às restrições de integridade do banco de dados.

Através de experimentação, a presença de conflitos de tradução de operações de edição XML para instruções de atualização no banco de dados pode variar conforme o tipo de operação. Portanto, um determindo tipo de operação informada no delta pode ter um significado diferente no BDR.

5.1. Contribuições da Dissertação

Conforme exposto na seção 1.1, as contribuições deste trabalho são:

- A definição de uma abordagem para propagar modificações de dados XML para BDR;
- A criação e atribuição de identificadores para os nodos XML, preservando a identidade do nodo, mesmo após modificações na estrutura do documento;
- A geração de chaves para os dados XML com base nos identificadores atribuídos aos nodos e no seu conceito no esquema relacional;
- A definição de equivalência entre as operações de modificação detectadas nos documentos XML em relação às operações de atualização sobre o BDR, na linguagem SQL.

Em relação a publicações, está sendo trabalhado um artigo com a descrição da abordagem proposta. A escrita de outro artigo também está idealizada para pós-defesa, tendo como foco o processo de definição e atribuição de chaves XML.

5.2. Trabalhos Futuros

Existem algumas propostas de melhorias e trabalhos futuros no tema de pesquisa, que são apresentados a seguir:

- O atual processo de manutenção do documento XML com as chaves geradas está limitado para atribuir ou deletar novas chaves. Ou seja, a cada nova chave gerada o documento XML precisa ser reconstruído totalmente, tornando o processo de manutenção oneroso. De forma a complementar a abordagem existente, a definição de uma estratégia automatizada para manutenção do documento com as chaves geradas é necessária para tornar o processo mais eficiente;
- No sentido de otimizar o esquema de banco de dados gerado a partir dos documentos XML, técnicas baseadas na refatoração de banco de dados podem permitir um esquema melhor otimizado. Além disso, técnicas baseadas em normalização, eliminação de redundâncias e replicação de dados poderão ser aplicadas para garantir uma base de dados mais refinada;
- A atual abordagem para geração de chaves está limitada ao rastreamento da tupla nos documentos XML através da restrição de chave primária. Para permitir a propagação de modificações para tabelas associadas, a proposta para geração das chaves deve incluir, também, as restrições de chave estrangeira;
- De maneira a permitir a propagação de modificações no BDR para os documentos XML, estudos podem ser direcionados à tratativa do clássico problema de manutenção de visões XML;
- Para permitir experimentos mais abrangentes, novas fontes de dados heterogêneas podem ser usadas para avaliar os resultados obtidos.

Apêndice A – Exemplo de Documento de Mapeamento

Exemplo de documento de mapeamento representativo dos conceitos entre os esquemas XML, a ontologia e o modelo lógico relacional, gerado pelo componente CMap. Partindo do elemento raiz `<mapeamento>` (linha 2), o atributo `nome` do elemento `<conceito>` (linha 3) referencia o conceito da ontologia. A partir disso, o elemento `<fonte>` refere-se às fontes de dados XML identificadas pelo atributo `id` (linha 4) relativos ao conceito da ontologia. O elemento `<relacional>` (linha 10) informa o `<tipo>` (linha 151) e `<nome>` (linha 152) para os conceitos não-léxicos – equivalente a tabela, adicionalmente, os elementos `<tabela>` (linha 13) e `<domínio>` (linha 14) correspondem a conceitos léxicos – colunas da tabela.

```
 1.<?xml version="1.0" encoding="UTF-8"?>
 2.<mapping>
 3. <concept name="cod_book">
 4.  <source id="Doc_A.xml">
 5.   <xpath>/bibliography/book/cod_book</xpath>
 6.  </source>
 7.  <source id="Doc_B.xml">
 8.   <xpath>/bibliography/book/cod_book</xpath>
 9.  </source>
10.  <relational>
11.   <type>column</type>
12.   <name>cod_book</name>
13.   <table>book</table>
14.   <domain>integer</domain>
15.  </relational>
16. </concept>
17. <concept name="edition">
18.  <source id="Doc_A.xml">
19.   <xpath>/bibliography/book/edition</xpath>
20.  </source>
21.  <source id="Doc_B.xml">
22.   <xpath>/bibliography/book/edition</xpath>
23.  </source>
24.  <relational>
25.   <type>column</type>
26.   <name>edition</name>
27.   <table>book</table>
28.   <domain>string</domain>
29.  </relational>
30. </concept>
31. <concept name="last">
32.  <source id="Doc_A.xml">
33.   <xpath>/bibliography/book/author/names/last</xpath>
34.  </source>
35.  <source id="Doc_B.xml"/>
36.  <relational>
37.   <type>column</type>
```

```
38.    <name>last</name>
39.    <table>names</table>
40.    <domain>string</domain>
41.   </relational>
42. </concept>
43. <concept name="instituition">
44.   <source id="Doc_A.xml">
45.    <xpath>/bibliography/book/author/instituition</xpath>
46.   </source>
47.   <source id="Doc_B.xml"/>
48.   <relational>
49.    <type>column</type>
50.    <name>instituition</name>
51.    <table>author</table>
52.    <domain>string</domain>
53.   </relational>
54. </concept>
55. <concept name="pages">
56.   <source id="Doc_A.xml"/>
57.   <source id="Doc_B.xml">
58.    <xpath>/bibliography/book/pages</xpath>
59.   </source>
60.   <relational>
61.    <type>column</type>
62.    <name>pages</name>
63.    <table>book</table>
64.    <domain>integer</domain>
65.   </relational>
66. </concept>
67. <concept name="cod_names">
68.   <source id="Doc_A.xml">
69.    <xpath>/bibliography/book/author/names/cod_names</xpath>
70.   </source>
71.   <source id="Doc_B.xml"/>
72.   <relational>
73.    <type>column</type>
74.    <name>cod_names</name>
75.    <table>names</table>
76.    <domain>integer</domain>
77.   </relational>
78. </concept>
79. <concept name="bibliography">
80.   <source id="Doc_A.xml">
81.    <xpath>/bibliography</xpath>
82.   </source>
83.   <source id="Doc_B.xml">
84.    <xpath>/bibliography</xpath>
85.   </source>
86.   <relational>
87.    <type>table</type>
88.    <name>bibliography</name>
89.   </relational>
90. </concept>
91. <concept name="publisher">
92.   <source id="Doc_A.xml">
93.    <xpath>/bibliography/book/publisher</xpath>
94.   </source>
95.   <source id="Doc_B.xml">
96.    <xpath>/bibliography/book/publisher</xpath>
97.   </source>
98.   <relational>
99.    <type>column</type>
100.   <name>publisher</name>
101.   <table>book</table>
```

```
102.    <domain>string</domain>
103.   </relational>
104.  </concept>
105.  <concept name="author">
106.   <source id="Doc_A.xml">
107.    <xpath>/bibliography/book/author</xpath>
108.   </source>
109.   <source id="Doc_B.xml">
110.    <xpath>/bibliography/book/writer</xpath>
111.   </source>
112.   <relational>
113.    <type>table</type>
114.    <name>author</name>
115.   </relational>
116.  </concept>
117.  <concept name="title">
118.   <source id="Doc_A.xml">
119.    <xpath>/bibliography/book/title</xpath>
120.   </source>
121.   <source id="Doc_B.xml">
122.    <xpath>/bibliography/book/title</xpath>
123.   </source>
124.   <relational>
125.    <type>column</type>
126.    <name>title</name>
127.    <table>book</table>
128.    <domain>string</domain>
129.   </relational>
130.  </concept>
131.  <concept name="price">
132.   <source id="Doc_A.xml">
133.    <xpath>/bibliography/book/price</xpath>
134.   </source>
135.   <source id="Doc_B.xml">
136.    <xpath>/bibliography/book/price</xpath>
137.   </source>
138.   <relational>
139.    <type>column</type>
140.    <name>price</name>
141.    <table>book</table>
142.    <domain>float</domain>
143.   </relational>
144.  </concept>
145.  <concept name="names">
146.   <source id="Doc_A.xml">
147.    <xpath>/bibliography/book/author/names</xpath>
148.   </source>
149.   <source id="Doc_B.xml"/>
150.   <relational>
151.    <type>table</type>
152.    <name>names</name>
153.   </relational>
154.  </concept>
155.  <concept name="name">
156.   <source id="Doc_A.xml"/>
157.   <source id="Doc_B.xml">
158.    <xpath>/bibliography/book/writer/name</xpath>
159.   </source>
160.   <relational>
161.    <type>column</type>
162.    <name>name</name>
163.    <table>author</table>
163.    <domain>string</domain>
164.   </relational>
```

```
165. </concept>
167. <concept name="book">
168.  <source id="Doc_A.xml">
169.   <xpath>/bibliography/book</xpath>
170.  </source>
171.  <source id="Doc_B.xml">
172.   <xpath>/bibliography/book</xpath>
173.  </source>
174.  <relational>
175.   <type>table</type>
176.   <name>book</name>
177.  </relational>
178. </concept>
179. <concept name="cod_author">
180.  <source id="Doc_A.xml">
181.   <xpath>/bibliography/book/author/cod_author</xpath>
182.  </source>
183.  <source id="Doc_B.xml">
184.   <xpath>/bibliography/book/writer/cod_author</xpath>
185.  </source>
186.  <relational>
187.   <type>column</type>
188.   <name>cod_author</name>
189.   <table>author</table>
190.   <domain>integer</domain>
191.  </relational>
192. </concept>
193. <concept name="year">
194.  <source id="Doc_A.xml">
195.   <xpath>/bibliography/book/year</xpath>
196.  </source>
197.  <source id="Doc_B.xml">
198.   <xpath>/bibliography/book/year</xpath>
199.  </source>
200.  <relational>
201.   <type>column</type>
202.   <name>year</name>
203.   <table>book</table>
204.   <domain>integer</domain>
205.  </relational>
206. </concept>
207. <concept name="first">
208.  <source id="Doc_A.xml">
209.   <xpath>/bibliography/book/author/names/first</xpath>
210.  </source>
211.  <source id="Doc_B.xml"/>
212.  <relational>
213.   <type>column</type>
214.   <name>first</name>
215.   <table>names</table>
216.   <domain>string</domain>
217.  </relational>
218. </concept>
219.</mapping>
```

Apêndice B – Exemplo de Documento de Mapeamento dos Tokens

Exemplo de documento de mapeamento dos *tokens* que compõe a estrutura do documento XML representado graficamente na Figura 2.6. Partindo do elemento raiz `<mapeamento>` (linha 2), o `id` do elemento documento informa o identificador único da `fonte` (linha 6). A partir disso, o rótulo identificador do *token* está identificado pelos atributos `id`, `left`, `parent` e `right` (todos na linha 6) do elemento `<token>`. Cada *token* é identificado pelo atributo `nome` e tem a expressão de caminho `<expressão>` (linha 8) que aponta sua localização na estrutura do documento. Além disso, as propriedades do *token*, que são de interesse no processo de reconstrução do documento XML, são informadas no elemento `<propriedades>` (linha 10).

```
 1.<?xml version="1.0" encoding="UTF-8"?>
 2.<mapping>
 3. <document id="1">
 4.  <source name="Doc_A.xml"/>
 5.  <maxtokenid>23</maxtokenid>
 6.  <token id="1" left="" parent="" right="">
 7.   <source name="book">
 8.    <xpath>/book</xpath>
 9.   </source>
10.   <properties>
11.    <type>element</type>
12.   </properties>
13.  </token>
14.  <token id="2" left="" parent="1" right="">
15.   <source name="year">
16.    <xpath>/book/year</xpath>
17.   </source>
18.   <properties>
19.    <value>1999</value>
20.    <type>attribute</type>
21.   </properties>
22.  </token>
23.  <token id="3" left="" parent="1" right="4">
24.   <source name="title">
25.    <xpath>/book/title</xpath>
26.   </source>
27.   <properties>
28.    <value>The Economics of Technology and Content for Digital TV</value>
29.    <type>element</type>
30.   </properties>
31.  </token>
32.  <token id="4" left="3" parent="1" right="9">
33.   <source name="author">
34.    <xpath>/book/author</xpath>
35.   </source>
36.   <properties>
37.    <type>element</type>
```

```
 38.   </properties>
 39.  </token>
 40.  <token id="5" left="" parent="4" right="10">
 41.   <source name="names">
 42.    <xpath>/book/names</xpath>
 43.   </source>
 44.   <properties>
 45.    <type>element</type>
 46.   </properties>
 47.  </token>
 48.  <token id="6" left="" parent="5" right="7">
 49.   <source name="last">
 50.    <xpath>/book/last</xpath>
 51.   </source>
 52.   <properties>
 53.    <value>Gerbarg</value>
 54.    <type>element</type>
 55.   </properties>
 56.  </token>
 57.  <token id="7" left="6" parent="5" right="">
 58.   <source name="first">
 59.    <xpath>/book/first</xpath>
 60.   </source>
 61.   <properties>
 62.    <value>Darcy</value>
 63.    <type>element</type>
 64.   </properties>
 65.  </token>
 66.  <token id="8" left="4" parent="1" right="11">
 67.   <source name="publisher">
 68.    <xpath>/book/publisher</xpath>
 69.   </source>
 70.   <properties>
 71.    <value>Springer</value>
 72.    <type>element</type>
 73.   </properties>
 74.  </token>
 75.  <token id="9" left="11" parent="1" right="">
 76.   <source name="price">
 77.    <xpath>/book/price</xpath>
 78.   </source>
 79.   <properties>
 80.    <value>223.48</value>
 81.    <type>element</type>
 82.   </properties>
 83.  </token>
 84.  <token id="10" left="5" parent="4" right="">
 85.   <source name="instituition">
 86.    <xpath>/book/instituition</xpath>
 87.   </source>
 88.   <properties>
 89.    <value>CITI</value>
 90.    <type>element</type>
 91.   </properties>
 92.  </token>
 93.  <token id="11" left="8" parent="1" right="9">
 94.   <source name="edition">
 95.    <xpath>/book/edition</xpath>
 96.   </source>
 97.   <properties>
 98.    <value>1999 edition</value>
 99.    <type>element</type>
100.   </properties>
101.  </token>
```

```
<token id="12" left="1" parent="" right="">
 <source name="book">
  <xpath>/book</xpath>
 </source>
 <properties>
  <type>element</type>
 </properties>
</token>
<token id="13" left="" parent="12" right="14">
 <source name="title">
  <xpath>/book/title</xpath>
 </source>
 <properties>
  <value>Advanced Programming in the Unix environment</value>
  <type>element</type>
 </properties>
</token>
<token id="14" left="13" parent="12" right="21">
 <source name="author">
  <xpath>/book/author</xpath>
 </source>
 <properties>
  <type>element</type>
 </properties>
</token>
<token id="15" left="" parent="14" right="18">
 <source name="names">
  <xpath>/book/author/names</xpath>
 </source>
 <properties>
  <type>element</type>
 </properties>
</token>
<token id="16" left="" parent="15" right="17">
 <source name="last">
  <xpath>/book/author/names/last</xpath>
 </source>
 <properties>
  <value>Stevens</value>
  <type>element</type>
 </properties>
</token>
<token id="17" left="16" parent="15" right="">
 <source name="first">
  <xpath>/book/author/names/first</xpath>
 </source>
 <properties>
  <value>W</value>
  <type>element</type>
 </properties>
</token>
<token id="18" left="15" parent="12" right="">
 <source name="names">
  <xpath>/book/author/names</xpath>
 </source>
 <properties>
  <type>element</type>
 </properties>
</token>
<token id="19" left="" parent="18" right="14">
 <source name="last">
  <xpath>/book/author/names/last</xpath>
 </source>
 <properties>
```

```
166.    <type>element</type>
167.   </properties>
168.  </token>
169.  <token id="20" left="19" parent="18" right="">
170.   <source name="first">
171.    <xpath>/book/author/names/first</xpath>
172.   </source>
173.   <properties>
174.    <value>Stephen</value>
175.    <type>element</type>
176.   </properties>
177.  </token>
178.  <token id="21" left="14" parent="12" right="22">
179.   <source name="publisher">
180.    <xpath>/book/publisher</xpath>
181.   </source>
182.   <properties>
183.    <value>Addison-Wesley</value>
184.    <type>element</type>
185.   </properties>
186.  </token>
187.  <token id="22" left="21" parent="12" right="23">
188.   <source name="edition">
189.    <xpath>/book/edition</xpath>
190.   </source>
191.   <properties>
192.    <value>2</value>
193.    <type>element</type>
194.   </properties>
195.  </token>
196.  <token id="23" left="22" parent="12" right="">
197.   <source name="price">
198.    <xpath>/book/price</xpath>
199.   </source>
200.   <properties>
201.    <value>53.58</value>
202.    <type>element</type>
203.   </properties>
204.  </token>
205. </document>
206.</mapping>
```

Apêndice C – Exemplo Esquema Lógico Representado em XML

Exemplo de documento de representação do modelo lógico relacional de banco de dados gerado a partir da ontologia representativa do esquema relacional dos documentos A e B da Figura 2.5, respectivamente. Tal artefato foi gerado pelo componente OntoRel juntamente com o *script* SQL de criação das tabelas.

```
 1.<?xml version="1.0" encoding="UTF-8"?>
 2.<tabelas>
 3. <tabela>
 4.  <nome>author</nome>
 5.  <coluna>
 6.   <nome>cod_author</nome>
 7.   <tipo>integer</tipo>
 8.   <obrigatoria>true</obrigatoria>
 9.  </coluna>
10.  <coluna>
11.   <nome>instituition</nome>
12.   <tipo>string</tipo>
13.   <obrigatoria>false</obrigatoria>
14.  </coluna>
15.  <coluna>
16.   <nome>name</nome>
17.   <tipo>string</tipo>
18.    <obrigatoria>false</obrigatoria>
19.  </coluna>
20.  <restricao>
21.   <tipo>chave_primaria</tipo>
22.   <coluna>cod_author</coluna>
23.  </restricao>
24. </tabela>
25. <tabela>
26.  <nome>book</nome>
27.  <coluna>
28.   <nome>cod_book</nome>
29.   <tipo>integer</tipo>
30.   <obrigatoria>true</obrigatoria>
31.  </coluna>
32.  <coluna>
33.   <nome>pages</nome>
34.   <tipo>integer</tipo>
35.   <obrigatoria>false</obrigatoria>
36.  </coluna>
37.  <coluna>
38.   <nome>year</nome>
39.    <tipo>integer</tipo>
40.    <obrigatoria>true</obrigatoria>
41.  </coluna>
42.  <coluna>
43.   <nome>title</nome>
44.   <tipo>string</tipo>
45.   <obrigatoria>true</obrigatoria>
46.  </coluna>
```

```
 47.  <coluna>
 48.   <nome>publisher</nome>
 49.   <tipo>string</tipo>
 50.   <obrigatoria>true</obrigatoria>
 51.  </coluna>
 52.  <coluna>
 53.   <nome>price</nome>
 54.   <tipo>float</tipo>
 55.   <obrigatoria>true</obrigatoria>
 56.  </coluna>
 57.  <coluna>
 58.   <nome>edition</nome>
 59.   <tipo>string</tipo>
 60.   <obrigatoria>true</obrigatoria>
 61.  </coluna>
 62.  <coluna>
 63.   <nome>cod_author</nome>
 64.   <tipo>integer</tipo>
 65.   <obrigatoria>true</obrigatoria>
 66.  </coluna>
 67.  <restricao>
 68.   <tipo>chave_primaria</tipo>
 69.   <coluna>cod_book</coluna>
 70.  </restricao>
 71.  <restricao>
 72.   <tipo>chave_estrangeira</tipo>
 73.   <coluna>cod_author</coluna>
 74.   <referencia>author</referencia>
 75.  </restricao>
 76. </tabela>
 77. <tabela>
 78.  <nome>names</nome>
 79.  <coluna>
 80.   <nome>cod_names</nome>
 81.   <tipo>integer</tipo>
 82.   <obrigatoria>true</obrigatoria>
 83.  </coluna>
 84.  <coluna>
 85.   <nome>last</nome>
 86.   <tipo>string</tipo>
 87.   <obrigatoria>true</obrigatoria>
 88.  </coluna>
 89.  <coluna>
 90.   <nome>first</nome>
 91.   <tipo>string</tipo>
 92.   <obrigatoria>true</obrigatoria>
 93.  </coluna>
 94.  <restricao>
 95.   <tipo>chave_primaria</tipo>
 96.   <coluna>cod_names</coluna>
 97.  </restricao>
 98. </tabela>
 99. <tabela>
100.  <nome>authornames</nome>
101.  <coluna>
102.   <nome>cod_author</nome>
103.   <tipo>integer</tipo>
104.   <obrigatoria>true</obrigatoria>
105.  </coluna>
106.  <coluna>
107.   <nome>cod_names</nome>
108.   <tipo>integer</tipo>
109.   <obrigatoria>true</obrigatoria>
110.  </coluna>
```

```
111.  <restricao>
112.   <tipo>chave_primaria_composta</tipo>
113.   <coluna>cod_author,cod_names</coluna>
114.  </restricao>
115.  <restricao>
116.   <tipo>chave_estrangeira</tipo>
117.   <coluna>cod_author</coluna>
118.   <referencia>author</referencia>
119.  </restricao>
120.  <restricao>
121.   <tipo>chave_estrangeira</tipo>
122.   <coluna>cod_names</coluna>
123.   <referencia>names</referencia>
124.  </restricao>
125. </tabela>
126.</tabelas>
```

Apêndice D – Exemplo de Arquivo Delta gerado pelo Algoritmo X-Diff

Exemplo de arquivo delta gerado pelo algoritmo X-Diff a partir da comparação do documento original e modificado da Figura 3.12.

```
 1.<?xml version="1.0" encoding="UTF-8"?>
 2.<bibliography>
 3. <book year="1999">
 4.  <title>The Economics of Technology and Content for Digital TV</title>
 5.  <author>
 6.   <names>
 7.    <last>Gerbarg</last>
 8.    <first>Darcy</first>
 9.   </names>
10.   <instituition><?INSERT instituition?>CITI</instituition>
11.  </author>
12.  <publisher>Springer</publisher>
13.  <edition><?INSERT edition?>1999 edition</edition>
14.  <price>223.48<?UPDATE FROM “129.00”?></price>
15. </book>
16. <book year="2005">
17.  <title>Advanced Programming in the Unix environment</title>
18.  <author>
19.   <names>
20.    <last>Stevens</last>
21.    <first>W.</first>
22.   </names>
23.   <names>
24.    <last>Rago</last>
25.    <first>Stephen</first>
26.   </names>
27.  </author>
28.   <publisher>Addison-Wesley</publisher>
29.   <edition>2</edition>
30.   <price>53.58</price>
31. </book>
32.</bibliography>
```

REFERÊNCIAS BIBLIOGRÁFICAS

AL-EKRAM, R.; ADMA, A.; BAYSAL, O. diffX: an algorithm to detect changes in multi-version XML documents. In Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research (CASCON '05), James R. Cordy, Anatol W. Kark, and Darlene A. Stewart (Eds.). IBM Press 1-11. 2005.

AVELAR, F. M.; SACCOL, D. B.; PIVETA, E. K. An ontology-based approach for storing XML data into relational databases. In: International Conference on Software Engineering and Knowledge Engineering (SEKE ‘12), pp.438-443 Redwood City, San Francisco Bay. 2012.

BRAGANHOLO, V., DAVIDSON, S., HEUSER, C. Pataxó: a framework to allow updates through xml views. ACM Transactions on Database Systems, (TODS ’06), New York, v. 31, n. 3. 2006.

CHEN, H.; LIAO, H. A Comparative Study of View Update Problem. *In* Proceedings of the 2010 International Conference on Data Storage and Data Engineering (DSDE '10). IEEE Computer Society, Washington, DC, USA, 83-89. 2010.

CHOI, B. et al. Updating recursive XML views of relations. Journal of Computer Science and Technology. Pp. 516-537. 2008.

COBÉNA, G.; ABITEBOUL, S.; MARIAN, A. Detecting Changes in XML Documents. In Proceedings of the 18th International Conference on Data Engineering (ICDE '02). IEEE Computer Society, Washington, DC, USA, 41. 2002.

DAYAL, U.; BERNSTEIN, A. On the correct translation of update operations on relational views. ACM Transactions on Database System, (TODS ’82), v.7 n.3, pp.381-416. 1982.

DBLP. Digital Bibliography & Library Project. Disponível em: <http://dblp.uni-trier.de/>. Acesso em 08 de abr. 2013.

DIETZ, P.F. “Maintaining Order in a Linked List,” Proceedings of the Anual ACM Symposium on Theory of Computing. New York, USA. 1982.

DWEIB, I. M. Storing XML documents in Relational Database using Multi Dimension Links. International Workshops in Electrical Engineering, Koc University, Istanbul/Turkey. pp. 277-284. 2013.

DWEIB, I.; AWADI, A.; LU, J. “MAXDOR: Mapping XML Document into Relational Database,” The Open Information System Journal, vol. 3, pp. 108-122, June 2009.

FEGARAS, L.. "Propagating updates through XML views using lineage tracing," Data Engineering (ICDE), 2010 IEEE 26th International Conference on , pp.309-320, 1-6 March 2010.

FLORESCU, D.; KOSSMANN, D. Storing and querying XML data using an RDBMS. IEEE Data Engineering Bulletin, pp. 27-34, 1999.

FONTAINE, R. L. A delta format for xml: Identifying changes in xml files and representing the changes in xml. In: XML Europe, 2001. Proceedings... Berlin, Germany: [s.n.], 2001.

GUPTA, A.; MUMICK, I. S. Maintenance of materialized views: Problems, techniques and applications. IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and DataWarehousing, pp.3-18. 1995.

HEGARET, P. L.; WHITMER, R.; WOOD, L. Document object model (DOM). 2006. Disponível em: <http://www.w3.org/DOM/>. Acesso em 08 de abr. 2014.

LEONARDI, E.; BHOWMICK, S. S. Detecting Changes on Unordered XML Documents Using Relational Databases: a Schema-conscious Approach. In**:** ACM International Conference on Information and Knowledge Management, CIKM, 2005, New York, USA. Proceedings… New York: ACM Press, pp. 509–516. 2005.

LI, C.; LING, T.W.; HU, M. "Efficient Updates in Dynamic XML Data: From Binary String to Quaternary String," Int'l J. Very Large Data Bases, pp. 573-601, 2008.

LI, Q.; MOON, B. "Indexing and Querying XML Data for Regular Path Expressions," Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01), pp.361-370. 2001.

LOOSE, M. B. ; SACCOL, D. B. Translating XML Queries into Equivalent SQL Statements. In: IADIS International Conference on WWW/Internet. Madrid, Espanha. 2012.

MELLO, M. R. OntoGen: Uma Ferramenta para Integração de Esquemas XML. Trabalho de Conclusão – Curso de Ciência da Computação, UFRGS, Porto Alegre. 2007.

MICROSOFT. The XML Diff and Patch GUI Tool. Disponível em: <http://msdn.microsoft.com/en-us/library/aa302295.aspx> Acesso em 09 abr. 2014.

MONSELL. Merging XML Changes with DeltaXML. MONSELL. EDM Ltd. Disponível em <http://www.deltaxml.com/>. Acesso em 08 de abr. 2014.

NEGRINI, D. et al. "CMAP: Tracking equivalences from XML documents to relational schemas," Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on , pp.332-339,. 2012.

O'NEIL, P. et al. "ORDPATHs: Insert-Friendly XML Node Labels," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04), New York, USA. pp. 903-908. 2004.

ORACLE. Using XMLType. ORACLE, 2002. Disponível em: <http://docs.oracle.com/cd/B10501_01/appdev.920/a96620/xdb04cre.htm>. Acesso em 08 de abr. 2014.

SACCOL, D. B.; ANDRADE, T. C.; PIVETA, E. K. Mapping OWL Ontologies to Relational Schemas (to appear). IEEE IRI - International Conference on Information Reuse and Integration, Las Vegas, USA. 2011.

SACCOL, D. et al. Managing application domanins in P2P systems. Information Reuse and Integration, IRI. IEEE International Conference on, pp. 451-456. 2008.

SCHUCH. Arquivos XML Experimentais. SCHUCH, 2014. Disponível em: <https://sites.google.com/site/arquivosexperimentais/>. Acesso em 8 de abr. 2014.

SOLTAN, S.; RAHGOZAR, M. "A Clustering-based Scheme for Labelling XML Trees". IJCSNS International Journal of Computer Science and Network Security, pp. 84-89. 2006.

TATARINOV, I. et al. Storing and Querying Ordered XML using a Relational Database System. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of data (SIGMOD '02). pp. 204-215. 2002.

TORSTEN, G.; KEULEN, M. V.; JENS, T. "Accelerating XPath Evaluation in Any RDBMS". ACM Transactions on Database Systems, pp. 91-131. 2004.

VARGAS, A. Conflict Resolution and Delta Detection in Updates through XML views. In: DataX 2006 - EDBT Second International Workshop on Database Technologies for Handling XML Information on the Web, 2006, Munique, Alemanha. Current Trends in Database Technology EDBT 2006. Berlin Heidelberg : Springer, 2006. pp. 192-205.

VYSNIAUSKAS, E; NEMURAITE, L. Transforming Ontology Representation from OWL to Relational Database. Information Technology And Control, Kaunas, Technologija, pp. 333-343. 2006.

WANG, L.; RUNDESTEINER, E.; MANI, M. Updating XML views published over relational databases: towards the existence of a correct update mapping. *Data &* Knowledge Engineering, pp. 263-298. 2006.

WANG, Y.; DEWITT, D.; CAI, J. X-Diff: An Effective Change Detection Algorithm for XML Documents, pp. 519-530, Umeshwar Dayal, Krithi Ramamritham, T. M. Vijayaraman (Ed.), 19th International Conference on Data Engineering, IEEE Computer Society Press, Bangalore, India, March 2003.

WEIGEL, F.; SCHULZ, K.U.; MEUSS, H. "The BIRD Numbering Scheme for XML and Tree Databases - Deciding and Reconstructing Tree Relations Using Efficient Arithmetic Operations". In Proceedings of the Third international conference on Database and XML Technologies (XSysm '05), pp. 49-67. 2005.

XU, L. et al. "DDE: From Dewey to a Fully Dynamic XML Labeling Scheme," Proc. 35th ACM SIGMOD Int'l Conf. Management of Data, pp. 719-730. 2009.

ZHANG, C. et al. "On Supporting Containment Queries in Relational Database Management Systems," Proc. ACM SIGMOD Int'l Conf. Management of Data, 2001.

ZHANG, C. et al. Design and Implementation of Mapping Rules from OWL to Relational Database. In Proceedings of the WRI World Congress on Computer Science and Information Engineering, pp. 71-75, 2009.