UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Materialização de Visões XML

por
DEISE DE BRUM SACCOL

Dissertação submetida à avaliação, como
requisito parcial, para a obtenção do grau
de Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, dezembro de 2001

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Saccol, Deise de Brum

Materialização de Visões XML / por Deise de Brum Saccol. Porto Alegre: PPGC da UFRGS, 2001.

85 f:il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Heuser, Carlos Alberto.

1. Dados Semi-Estruturados. 2. Visões Materializadas. 3. XML. 4. Identificação de Instâncias XML 5. Atualização Incremental de Visões. I. Heuser, Carlos Alberto. II. Título.

"A partir de um determinado momento, este aluno tomará consciência de que nele e por ele se inicia um processo complexo de aprendizagem de ordem superior, em que a reflexão, o exercício da consciência crítica, o emprego do raciocínio, dos processos discursivos, da análise à síntese, da criatividade à demonstração, da inferência à verificação, constituem as atividades que se sobrepõem ao mero ato de ler e de ouvir para se informar. Tomará consciência de que terá que agir por autodeterminação, mesmo orientado, na assimilação do meio social, ao invés de simplesmente adaptar-se a ele..."

Délcio Vieira Salomon

Agradecimentos

Ter vencido mais esta etapa da vida faz-me refletir sobre alguns pontos. Os pensamentos levam-me a tempos longínquos... Lembro-me do primeiro dia de aula do primário, na pequena escola do interior de Santa Maria... Passam pelo Colégio onde meu pai estudou, quando ainda criança, e pela escola onde meus irmãos também já estiveram. Os pensamentos seguem... Vislumbro agora meus anos de segundo grau, trazendo-me à mente o antigo "Científico"!

Relembro uma das fases mais marcantes da minha vida, iniciada aos meados de março de 1995, na universidade. Tenho vivas todas as memórias e os acontecimentos maravilhosos daquela época. Sinto um pouco de melancolia.... Melancolia que se confunde no grande número de rostos que passam em minha mente... A amiga querida, o amigo sempre disposto a auxiliar, o professor sempre disposto a ouvir e a aconselhar...

E depois, chegam os tempos de mestrado... "Mas quanto orgulho estar cursando uma Pós-Graduação!" O nome pomposo assustava no início, mas ia tornando-se corriqueiro, à medida que os obstáculos iam sendo vencidos, e as vitórias, pouco a pouco, conquistadas.

Hoje, finalizando mais esta etapa, não posso deixar de citar alguns nomes que foram de fundamental importância nesta caminhada. O primeiro deles... Deus! Sem a sua força divina, nada disso seria possível. Pela sua mão estendida e pelos pés que andaram lado a lado comigo durante toda a minha vida, meu mais singelo agradecimento.

Em segundo lugar, meus pais! Minha eterna ausência, por estar vivendo em outra cidade, fez valorizar ainda mais o imenso amor que sinto por eles... Aos meus irmãos, que sempre me deram apoio e incentivo, mesmo à distância, nos momentos mais difíceis. E ao meu sobrinho Gugu, que tantas vezes fez-me sair de casa com lágrimas nos olhos, de saudade...

Em terceiro lugar, o mais sincero e carinhoso agradecimento aos meus verdadeiros amigos, e em especial, minhas amigas Ale, Luciana e Anelise... companheiras fiéis de RU... quanta saudade! Só vocês sabem o verdadeiro significado desta vitória, porque vocês estiveram juntas, dia a dia, comigo...Aos amigos Adrovane, Carina, Vanessa, Lialda, Renata, Adriana, Lica e Sílvia...esta vitória é nossa!

Em hipótese alguma, poderia deixar de citar os mais recentes amigos, conquistados em Palmas, TO. Ao Centro Universitário Luterano de Palmas, muito obrigada pelo apoio e incentivo. Em especial, aos queridos companheiros: Parcilene, Fabiano, Theresa, Augusto, Piveta, Leal, Cristina e Lilissanne, meus sinceros agradecimentos.

Também meu grande reconhecimento por aquele que deu-me a oportunidade de chegar até aqui... meu orientador, professor Heuser. Pelo seu sorriso, empatia e estímulos constantes, meu sincero muito obrigada. Sua competência, inteligência, profissionalismo e caráter fizeram-me acreditar que era possível seguir em frente, mesmo quando as adversidades da vida nos desencorajam e nos enfraquecem. À você, minha eterna amizade e reconhecimento.

Meu agradecimento a UFRGS, CAPES e IBM/Solectron, que deram-me a oportunidade e os recursos financeiros para finalizar esta caminhada. Também meu muito obrigada a todos aqueles, que de uma forma ou de outra, auxiliaram e colaboraram para a finalização desta etapa.

Sumário

Lista de Abreviaturas

BD	Banco de Dados
CWA	Closed World Assumption
DBMS	Data Base Management System
DTD	Document Type Definition
HTML	HyperText Markup Language
IDOC	Intelligent Document
OID	Object Identifier
OLAP	On-line Analytical Processing
OWA	Open World Assumption
RI	Recuperação de Informação
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SQL	Structured Query Language
W3C	Word Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Lista de Figuras

Lista de Tabelas

Resumo

A grande quantidade de dados eletrônicos disponível atualmente nem sempre pode ser representada com modelos tradicionais, principalmente devido à ausência de esquema no momento da criação destes dados. Neste sentido, modelos semi-estruturados têm sido propostos; uma das abordagens utilizadas é XML, uma linguagem para troca e representação deste tipo de informação. Entretanto, consultar dados semi-estruturados pode demandar processos de extração com alto custo. Uma das alternativas para solucionar este problema é a definição de visões sobre estes dados, e a posterior materialização destas informações.

O uso de visões materializadas para dados XML ainda é pouco explorado. Uma das abordagens que podem ser utilizadas é o uso de sistemas de gerenciamento de bancos de dados relacionais para o armazenamento das visões. Desse modo, informação semanticamente relacionada (informação acerca de um mesmo domínio, possivelmente representada em formatos diferentes) pode ser agrupada em uma única unidade lógica, facilitando o acesso a estes dados por parte do usuário, e introduzindo alguma estrutura nos dados semi-estruturados. Dessa maneira, o usuário final submete consultas diretamente sobre a visão materializada, evitando extrações contínuas de dados nas fontes XML.

A materialização de dados XML exige a definição de um repositório de dados para o armazenamento destas instâncias. Utilizando-se a abordagem relacional, é necessário definir um mecanismo para a geração do esquema lógico do banco de dados. Consultar os dados nas fontes XML exige a integração destas instâncias. Neste contexto, integrá-las significa identificar quais instâncias de dados representam o mesmo objeto do mundo real, bem como resolver ambigüidades de representação deste objeto. O problema de identificação de entidades em XML é mais complexo que em bases de dados estruturadas. Dados XML, como propostos originalmente, não carregam necessariamente a noção de chave primária ou identificador de objeto. Assim, é necessária a adoção de um mecanismo que faça a identificação das instâncias na integração destes dados.

Além disso, à medida que as fontes de dados XML sofrem alterações, a visão materializada deve ser atualizada, a fim de manter-se consistente com as fontes de dados. A manutenção deve propagar as alterações feitas nos dados XML para a visão materializada. Reprocessar todo o conteúdo da visão materializada é, na maioria das vezes, muito caro. Assim, é desejável propagar as mudanças incrementalmente, ou seja, processar apenas as alterações necessárias.

Neste sentido, o presente trabalho apresenta uma proposta de técnica para armazenamento de dados XML em um banco de dados relacional. A proposta utiliza ontologias para a geração do esquema lógico do banco de dados. O problema de integração de dados é mostrado. O foco principal do trabalho está na proposta de uma técnica de atribuição de identificadores a instâncias XML, baseada no uso de funções *Skolem* e no padrão *XPath*, proposto pelo W3C. Também é proposto um mecanismo para manutenção incremental deste banco, à medida que as fontes XML sofrem atualizações.

Palavras-chave: dados XML, visões relacionais materializadas, esquemas relacionais para dados XML, atualização incremental de visões materializadas, integração de dados XML

TITLE: "MATERIALIZATION OF XML VIEWS"

Abstract

The great amount of electronic data not always can be represented with traditional models, mainly due to the absence of a scheme at the moment of the creation of these data. In this direction, semistructured models have been considered; one of the approaches is XML, a language for exchanging and representing this type of information. However, to query semistructured data can demand expensive extraction processes. One of the alternatives to solve this problem is the definition of views over these data, and the materialization of this information. The use of materialized views for XML data is still little explored. One of the approaches that can be used is relational database management systems for storing these views. In this sense, semantic related information (information concerning to the same domain, possibly represented in different formats) can be grouped in a logical unit, making easier the access to these data and introducing some structure in semistructured data. In this way, the end users query over the materialized view, preventing continuous extractions of data in XML sources. The materialization of XML data demands the definition of a repository to store these instances. Using a relational database, it is necessary to define a mechanism for the generation of the logical scheme. To query the data in XML sources demands the integration of these instances. In this context, to integrate means to identify which instances of data represent the same object of the real world, as well as solving ambiguities of representation of this object. The problem of entities identification in XML is more complex than in structured databases. XML data, as originally considered, do not have the identification notion of primary key. Thus, it is necessary the adoption of a mechanism that identifies the instances at the moment of the data integration. Moreover, while the data sources are changing, the materialized view must be kept update, in order to remain itself consistent with the sources. The maintenance must propagate the changes made in XML data to the materialized view. Recompute all the content of the materialized view is, usually, very expensive. Thus, it is desirable to propagate the changes incrementally, which means to process only the necessary changes. In this direction, the present work presents a proposal of technique for storing XML data in a relational database. The proposal uses ontologies for generating the logical scheme. The problem of data integration is shown. The main focus of this work is to propose a technique to identify XML instances, based on the use of *Skolem* functions and *XPath*, proposed by W3C. Also, a mechanism for incremental maintenance of this database is considered.

Keywords: XML data, relational materialized views, relational schemas for XML data XML, incremental update of materialized views, integration of XML data

1 Introdução

A quantidade de dados disponível eletronicamente tem crescido muito nos últimos anos. Estes dados estão nos mais variados formatos, desde sistemas de arquivos totalmente desestruturados até informação armazenada em sistemas de bancos de dados convencionais. Grandes empresas gerenciam vários repositórios heterogêneos de dados e muitas aplicações requerem acesso a esta informação, armazenada em diferentes modelos e mecanismos de acesso.

Essa ampla variedade de informação nem sempre pode se representada com os modelos de dados relacionais, objeto-relacional ou orientado a objetos; exemplos são dados da WEB (bibliotecas digitais, documentação *on-line*, sites de comércio eletrônico, etc), para os quais geralmente não se conhece o esquema no momento em que são criados. Para suportar o acesso a este tipo de informação, modelos de dados semi-estruturados têm sido propostos. Uma definição objetiva de dados semi-estruturados é encontrada em [Abi97a]: dados semi-estruturados são dados que apresentam uma representação estrutural irregular, não sendo nem totalmente "crus" (sem nenhuma estrutura) nem estritamente tipados. Em alguns casos os dados possuem alguma estrutura (listas de referências retornadas por máquinas de pesquisa), em outros a estrutura está total ou parcialmente embutida no dado (documentos no formato de artigo) ou quase não apresentam informações descritivas (imagens).

XML vem sendo reconhecida como uma abordagem importante para representar dados semi-estruturados. No entanto, submeter consultas diretamente a fontes semi-estruturadas pode exigir um alto custo de acesso, demandando onerosos processos de extração. Em tratando-se deste tipo de dado, visões podem ser utilizadas para introduzir alguma estrutura nestas informações. Além disso, visões definidas sobre fontes de dados que sofrem atualizações pouco freqüentes podem ser materializadas em uma nova fonte de dados, garantindo melhoras de performance e maior facilidade no acesso às informações. Desse modo, consultas podem ser submetidas diretamente sobre a visão materializada, evitando repetições contínuas do processo de extração nas fontes semi-estruturadas, sempre que possível.

Visões materializadas relacionais para dados semi-estruturados têm sido apresentadas na literatura [DEU99, DEU99a, FLO99, FLO99a, SHA99]. Neste enfoque, o objetivo maior é aproveitar as funcionalidades e recursos oriundos do estágio de maturidade já atingidos pelos sistemas relacionais (como a facilidade para consultas através de SQL, por exemplo [SIL2000a]).

Vários tópicos têm sido pesquisados em relação à XML. Entretanto, enquanto o uso de visões materializadas sobre dados relacionais é bastante estudado, sua aplicação em dados semi-estruturados é uma área relativamente nova. Assim, este trabalho propõe uma técnica para materialização relacional de dados XML. A arquitetura apresentada é composta de dois módulos principais: geração do esquema lógico da visão materializada relacional (a partir de uma ontologia que descreve o domínio do problema das fontes de dados XML) e a integração, instanciação e manutenção incremental da visão materializada, à medida que os dados XML das fontes sofrem alterações.

O termo *Ontologia* já é conhecido e aplicado há bastante tempo na área da Filosofia, significando um “sujeito de existência” ou uma "contabilização sistemática da Existência" [MEL2000]. Na década de 90, este conceito passou a ser utilizado na Ciência da Computação. Recentemente, ontologias vêm sendo utilizadas nas áreas de BD e RI como suporte à interoperabilidade de fontes de dados distribuídas e heterogêneas. Do ponto de vista de BD, pode ser definida como uma especificação parcial de um domínio, descrevendo entidades, relações entre entidades e regras de integridade. No contexto deste trabalho, a ontologia é vista como um esquema conceitual que descreve o domínio das fontes de dados XML. A partir desta ontologia, gera-se o esquema lógico do banco de dados relacional.

A inserção de dados XML na visão materializada exige que se faça a integração das instâncias. No momento da integração dos dados XML, surge o problema de identificar se duas instâncias, vindas de fontes diferentes, modelam o mesmo objeto do mundo real. Dados XML, como propostos originalmente, não exigem identificadores; mesmo quando presentes, restringem-se a distinguir instâncias dentro de um único documento, não sendo aplicáveis à integração de diferentes documentos.

Muitas dos problemas encontrados na integração de dados da WEB são similares aos encontrados na integração de bancos de dados tradicionais. Porém, a integração de dados da WEB apresenta alguns problemas adicionais: o grande número de fontes de dados, a falta de metadados sobre estas fontes, o alto grau de autonomia das fontes de dados e a grande irregularidade na estrutura dos dados [SAL2001]. O problema de integração de dados XML é uma área recente de pesquisa.

Uma vez que a visão tenha sido materializada e os dados integrados, esta deve ser mantida atualizada à medida em que são feitas alterações nas fontes de dados. Este problema tem sido bastante estudado no modelo relacional. Uma solução trivial, mas geralmente de alto custo, é reprocessar os dados na visão materializada, através do processamento das consultas e conseqüentemente da extração dos dados que definem a visão. Bem mais interessante é propagar somente as alterações à visão materializada, processo este conhecido como manutenção incremental.

O trabalho está organizado da seguinte maneira: o capítulo 2 apresenta o estado da arte na materialização de dados XML, enfatizando a abordagem relacional. O capítulo 3 apresenta as principais abordagens encontradas na literatura para o tratamento de integração de instâncias, envolvendo a identificação propriamente dita e a resolução de conflitos nos valores de propriedades. A proposta do trabalho é mostrada no capítulo 4. Este capítulo apresenta as técnicas propostas para a geração do esquema lógico do banco de dados relacional a partir da ontologia, integração das instâncias XML e o mecanismo utilizado para a instanciação e manutenção incremental deste banco de dados. Conclusões e trabalhos futuros são apresentados no capítulo 5.

2 Materialização de Dados Semi-Estruturados

Este capítulo apresenta o estado da arte na materialização de dados semi-estruturados. A materialização de visões na abordagem relacional é descrita em maiores detalhes. As técnicas propostas na literatura para a geração de esquemas lógicos relacionais para o armazenamento de dados semi-estruturados também são apresentadas.

2.1 Visão Geral

Um dos objetivos da pesquisa em fontes semi-estruturadas é definir uma visão mais estruturada destes dados, de forma que os mesmos possam ser manipulados mais facilmente. Para tanto, são necessários mecanismos que identifiquem, recuperem e estruturem dados semi-estruturados relevantes. O processo que realiza essas tarefas pode ser chamado de extração de dados e as ferramentas que auxiliam nesse processo são chamadas extratores [MEL2000a]. A extração de dados normalmente transforma dados semi-estruturados de uma fonte de dados, como a WEB, para dados adequados a um modelo de dados; desse modo, os dados extraídos podem ser materializados em uma nova fonte de dados.

Há quatro abordagens possíveis para armazenar dados semi-estruturados [FLO99]. A primeira delas é construir um sistema de banco de dados de propósito específico. Exemplos de protótipos desta categoria são *Rufus* [SHO93], *Lore* [McH97] e *Strudel* [FER98]. Estes sistemas são projetados especificamente para armazenar e recuperar dados semi-estruturados, utilizando para isso estruturas e índices especiais e técnicas específicas de otimização de consultas.

A segunda abordagem consta em utilizar um sistema de banco de dados orientado a objetos. Nesta abordagem, são exploradas as amplas capacidades de modelagem de dados deste tipo de sistema. Um sistema comercial já implementado nesta categoria é o O2 [CHR94]. O projeto *Monet* também segue esta abordagem [ZWO99].

A terceira abordagem para o armazenamento de dados semi-estruturados consta na utilização de um sistema de banco de dados relacional (apresentado na seção 2.2). Neste caso, dados XML são mapeados para tabelas de um esquema relacional e as consultas realizadas sobre os dados semi-estruturados podem ser traduzidas para consultas SQL.

Uma última abordagem para o armazenamento de dados semi-estruturados consiste na utilização de arquivos texto. Esta técnica pode ser utilizada quando deseja-se apenas guardar as informações de dados semi-estruturados.

Das quatro abordagens apresentadas, pode-se salientar algumas considerações [FLO99]. Sistemas de propósito específico, teoricamente, deveriam ser bem aceitos e apresentar-se como a melhor alternativa. Entretanto, ainda deve-se levar um tempo relativamente longo para

que tais sistemas amadureçam e atinjam uma boa escalabilidade para grandes volumes de dados. Da mesma maneira, a geração atual de banco de dados orientados a objetos não é madura o suficiente para o processamento eficiente de consultas em bases de dados muito grandes. Em contrapartida, sistemas de bancos de dados relacionais apresentam boa escalabilidade, e possuem a vantagem de possibilitar a coexistência de dados XML e relacionais neste tipo de sistema. Com isso, é possível construir aplicações que utilizam ambos os tipos de dados com pouco esforço extra. Além disso, neste enfoque, o objetivo é aproveitar as funcionalidades e recursos oriundos do estágio de maturidade já atingidos pelos sistemas relacionais (como a facilidade para consultas através de SQL, por exemplo [SIL2000a]). O uso de sistemas relacionais para o armazenamento de dados semi-estruturados é descrito na seção a seguir.

2.2 Uso de Sistemas Gerenciadores de Bancos de Dados Relacionais para o Armazenamento de Visões XML

BDs tradicionais apresentam um esquema predefinido e uma estrutura homogênea a nível de atributos e tipos. Esse esquema serve de base para a inserção, consulta e outras operações sobre os dados. Em se tratando de dados semi-estruturados, cada ocorrência de dado pode ser heterogênea nos dois aspectos mencionados anteriormente. Assim, fica difícil existir um estrutura padrão para representá-los. Dada essa heterogeneidade, em geral a estrutura de um dado semi-estruturado está presente na própria descrição do dado, necessitando ser identificada e extraída. Essas tarefas são complexas, uma vez que a distinção entre esquema e dados nem sempre é clara, se comparar-se ocorrências diferentes de dados semanticamente iguais [MEL2000a].

Sistemas de gerenciamento de bancos de dados relacionais podem ser utilizados para o armazenamento de dados XML. Para otimizar o uso destes sistemas para o armazenamento de dados XML, trabalhos recentes têm se concentrado em modelos e algoritmos para extrair esquemas destes dados. Desse modo, algumas abordagens são apresentadas na literatura para a geração do esquema lógico relacional a partir de fontes de dados XML. As principais abordagens propõem: mapeamento de DTD, mapeamento específico entre modelos, mapeamento de atributos e valores, e o uso de ontologias. Estas abordagens são descritas a seguir.

2.2.1 Inferência do Esquema Lógico a partir da DTD dos Documentos

DTDs são utilizadas para descrever a estrutura de documentos XML e podem ser consideradas um esquema para estes documentos. A partir da DTD, pode-se definir a

seqüência dos elementos no documento, o conteúdo dos elementos e os atributos associados a cada elemento. Desse modo, pode-se realizar um processamento em cada tipo de elemento, inferindo como estes serão mapeados para tabelas de um banco de dados relacional.

Várias abordagens são encontradas na literatura para o mapeamento de uma DTD para um esquema lógico relacional. A maioria delas concentra-se em mapear elementos complexos para tabelas separadas, e elementos simples para colunas de tabelas. Algumas peculiaridades são apresentadas nestas propostas. A seguir, são mostradas duas destas propostas utilizadas para realizar o mapeamento de DTDs para esquemas relacionais.

A primeira destas abordagens consta na execução dos seguintes passos [BOU2001]:

- Para cada elemento complexo, cria-se uma tabela e uma coluna de chave primária.
- Para cada elemento com conteúdo misto, cria-se uma tabela separada para armazenar o PCDATA, ligado à tabela pai através da chave primária da tabela pai.
- Para cada atributo monovalorado de um elemento e para cada elemento filho simples de ocorrência única, cria-se uma coluna na tabela do elemento. Se o elemento ou o atributo do elemento for opcional, a coluna também é opcional.
- Para cada atributo multivalorado e para cada elemento filho simples de múltipla ocorrência, cria-se uma tabela separada para armazenar os valores, ligada à tabela pai através da chave primária da tabela pai.
- Para cada elemento filho complexo, liga-se a tabela do elemento pai à tabela do elemento filho com a chave primária da tabela pai.

O exemplo a seguir mostra como funciona este processo. Considerando a seguinte DTD [BOU2001]:

```
<!ELEMENT Order (OrderNum, Date, CustNum, Item*)>
<!ELEMENT OrderNum (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT CustNum (#PCDATA)>
<!ELEMENT Item (ItemNum, Quantity, Part)>
<!ELEMENT ItemNum (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Part (PartNum, Price)>
<!ELEMENT PartNum (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
```

Aplicando os passos anteriormente apresentados, tem-se o seguinte esquema lógico relacional gerado:

Order (<u>OrderPK</u>, OrderNum, Date, CustNum)
Item (<u>ItemPK</u>, ItemNum, Quantity, **OrderFK**)
Part (<u>PartPK</u>, PartNum, Price, **ItemFK**)

Onde: <u>OrderPK</u>, <u>ItemPK</u> e <u>PartPK</u> são chaves primárias; **OrderFK** e **ItemFK** são chaves estrangeiras.

Ao converter DTDs para esquemas relacionais, é bastante natural tentar mapear elementos na DTD para tabelas, e atributos de elementos para atributos das tabelas. Entretanto, este mapeamento direto pode causar uma grande fragmentação no documento, conforme pode ser visto no exemplo acima. Reconstruir um documento XML a partir de tabelas de um banco de dados relacional muito fragmentado pode ocasionar uma baixa performance. Desse modo, a técnica proposta a seguir visa diminuir essa fragmentação [SHA99]. Nesta proposta, os principais pontos considerados são:

- como tratar a complexidade permitida na especificação de elementos em uma DTD, levando em consideração a cardinalidade permitida nos elementos da DTD (opcional, obrigatório, multivalorado), elementos raiz, elementos folha, etc.

- como resolver os conflitos de mapeamento entre o esquema relacional (atributos e tabelas) e o aninhamento arbitrário de elementos na DTD;

- como tratar os atributos multivalorados e a recursão entre elementos: por exemplo, uma *monografia* possui um *editor* e um *editor* edita várias *monografias.*

Grande parte da complexidade das DTDs advém da especificação complexa do tipo de um elemento. Desse modo, é realizado um conjunto de transformações para simplificar a DTD. Essas transformações são de três tipos:

a) transformações de nivelamento, as quais convertem uma definição aninhada em uma representação plana. Por exemplo:

*(a, b)** é transformado em *a*, b**
(a, b)? é transformado em *a?, b?*
(a | b) é transformado em *a?, b?*

b) transformações de simplificação, as quais reduzem vários operadores unários para um único. Por exemplo:

*a*** é transformado em *a**
a?* é transformado em *a**
*a?** é transformado em *a**
a?? é transformado em *a?*

c) transformações de agrupamento, as quais agrupam sub-elementos que possuem o mesmo nome. Por exemplo:

..., a,...,a*, ...* é transformado em *a*, ...*

..., a,...,a?, ...* é transformado em *a*, ...*
..., a?,...,a, ...* é transformado em *a*, ...*
..., a?,...,a?, ... é transformado em *a*, ...*
..., a,...,a, ... é transformado em *a*, ...*

d) além disso, todos os operadores + são transformados em *.

Assim, um elemento definido como

<!ELEMENT a ((b | c |e)?, (e? | (f?, (b,b)*))*)>

seria transformado para <!ELEMENT a (b*, c?, e*, f*)>.

Estas transformações preservam a semântica de: *um* ou *vários* e *nulos* ou *não nulos*. Convém salientar que algumas informações de ordem relativa de elementos são perdidas nestas transformações, mas esse problema pode ser solucionado através da criação de um atributo *posição* nas tuplas representativas destes elementos.

Para a criação das tabelas, é criada uma estrutura de grafo, chamada de *grafo da DTD*. Basicamente, este grafo representa a estrutura da DTD: seus nós são elementos, atributos e operadores da DTD. Como exemplo, considere a seguinte DTD [SHA99]:

```
<!ELEMENT book (booktitle, author)>
<!ELEMENT article (title, author*, contactauthor?)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID IDREF #IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor (monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, address)>
<!ATTLIST author authorid ID #REQUIRED>
<!ELEMENT name (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT address (#PCDATA)>
```

O grafo da DTD é mostrado na FIGURA 1.

Dado o grafo da DTD, são criadas as tabelas do esquema relacional. A geração do esquema é baseada em certos critérios, como: número de arcos que chegam em um determinado elemento, existência de recursividade entre elementos, cardinalidade deste elemento (opcional, obrigatório, multivalorado), entre outros. A técnica objetiva garantir que um nó elemento seja representado em exatamente uma tabela, compartilhando nós comuns.

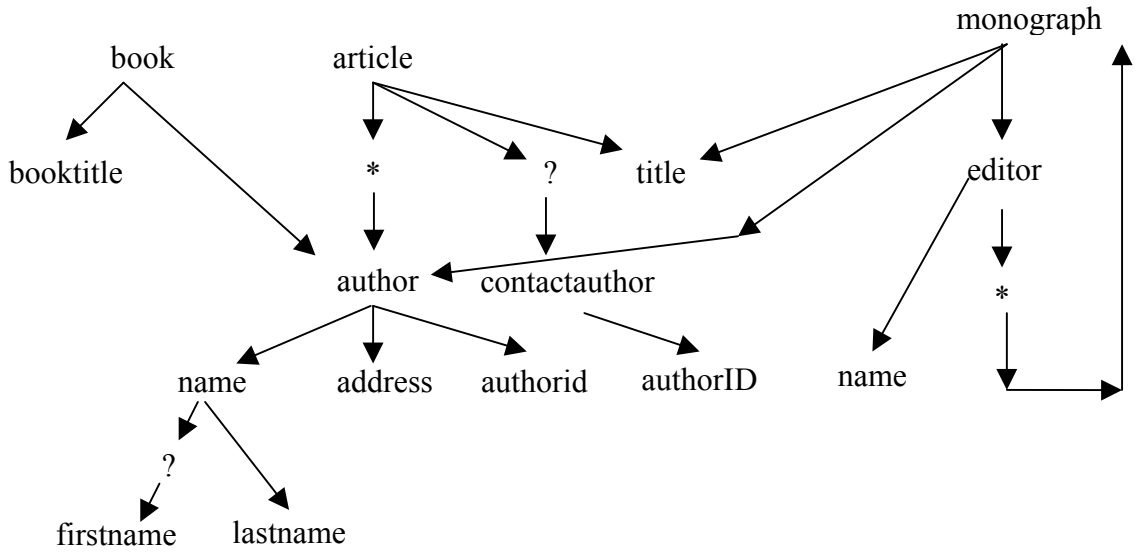

FIGURA 1 - Grafo da DTD [SHA99]

A idéia básica é criar tabelas para todos os elementos da DTD que não tenham algum nó apontando-os (*book, article*). Elementos com mais de um nó apontando-os, geram tabelas quando são recursivos. Todos os elementos alcançáveis por um nó * (independente do número de nós que apontam para ele: *author, monograph*) também geram tabelas separadas. Elementos apontados por um único nó são agrupados nas tabelas já existentes. Na existência de elementos mutuamente recursivos e alcançáveis por um único nó (como *monograph* e *editor*), um deles também deve gerar uma tabela separada. Os outros elementos restantes devem ser agrupados nas tabelas existentes.

Por exemplo, para o grafo da DTD da FIGURA 1, são geradas as seguintes tabelas:

Book (bookID, booktitle, firstname, lastname, address, authorid)
Article (articleID, **authorID**, title)
Monograph (monographID, title, firstname, lastname, address, authorid, editorName)
Author (authorID, firstname, lastname, address, id)

Onde: bookID, articleID,authorID, monographID, authorID são chaves primárias; **authorid** é chave estrangeira.

Os autores de um livro e de uma monografia são aninhados na tabela *livro* e *monografia*, respectivamente (já que há somente um autor para um livro e um autor para uma monografia). Entretanto, os autores de um artigo são armazenados na tabela separada do autor (já que pode haver mais de um autor para um artigo).

Como um segundo exemplo, considere a seguinte DTD:

```
<!ELEMENT Order (OrderNum, Date, CustNum, Item*)>
<!ELEMENT OrderNum (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT CustNum (#PCDATA)>
<!ELEMENT Item (ItemNum, Quantity, Part)>
<!ELEMENT ItemNum (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Part (PartNum, Price)>
<!ELEMENT PartNum (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
```

Aplicando os passos anteriormente apresentados, tem-se o seguinte esquema lógico relacional gerado:

Order (orderId, OrderNum, Date, CustNum)
Item (itemId, **orderId**, ItemNum, Quantity, PartNum, Price)

Onde: orderId e ItemId, orderId são chaves primárias; **orderId** é chave estrangeira.

2.2.2 Mapeamento Específico entre os Modelos Semi-Estruturado e Relacional

Esta abordagem propõe mapear o modelo de dados semi-estruturado para o modelo de dados relacional através de uma linguagem proprietária de consulta. O esquema relacional é definido e através da linguagem é feito o mapeamento da instância semi-estruturada para este esquema. A linguagem STORED, proposta por [DEU99a], segue esta abordagem. O mapeamento pode ser definido pelo programador ou derivado automaticamente para uma determinada instância de dados.

Uma consulta STORED consiste de um ou vários mapeamentos FROM WHERE STORE. A cláusula FROM consiste de um padrão único, ligando várias variáveis; cada cláusula FROM define uma variável chave única, que por definição, é a primeira variável no padrão. A cláusula STORE especifica como os valores serão ligados às variáveis armazenadas.

Por exemplo, considerando o seguinte documento [DEU99a]:

```
Audit: &o1
{taxpayer: &o24
    {name: &o41 “Gluschko”,
     address: &o34 {street: &105 “Tyuratam”}}
}
```

A consulta a seguir

```
FROM Audit.taxpayer: $X
{name: $N,
addr: {street: $S}}
STORE Taxpr1 ($X, $N, $S)
```

Realiza o seguinte armazenamento na tabela *Taxpr1*:

TABELA 1 - Tabela Taxpr1 Resultante do Mapeamento STORED

Oid	name	street
o24	Gluschko	Tyuratam

Além de poder ser definido manualmente pelo administrador, o mapeamento também pode ser derivado automaticamente [DEU99]. Este mapeamento automático leva em consideração alguns argumentos, especificados pelo usuário: número máximo de tabelas, número máximo de atributos por tabela, máximo espaço de disco, entre outros. Técnicas de mineração de dados são utilizadas para encontrar padrões na instância de dados e gerar automaticamente o esquema relacional e os mapeamentos expressos em STORED.

O algoritmo consiste de duas partes: mineração de dados e geração do armazenamento. Uma extensão do algoritmo *WL* [apud DEU99, 1999, p.4] é utilizada para encontrar padrões de armazenamento. Cada padrão define uma tabela. Desse modo, o algoritmo produz uma coleção de tabelas com um conjunto de atributos obrigatórios e opcionais. Um parâmetro *C*, definido pelo usuário, distingue entre coleções pequenas e grandes; deste modo, um atributo com menos de *C* ocorrências é considerado uma coleção pequena, e o algoritmo tenta produzir uma coluna para cada um destes atributos. Atributos com *C* ou mais ocorrências são representados em tabelas separadas.

2.2.3 Mapeamento de Atributos e Valores

Nesta abordagem, um documento XML é representado como um grafo rotulado dirigido: nós modelam objetos, arcos modelam atributos do objeto e folhas contém valores de dados (cadeias de caracteres); além disso, considera-se que cada objeto possui um identificador único. Vários mapeamentos podem ser utilizados para armazenar um documento XML em um banco de dados relacional [FLO99].

O ponto de partida desta abordagem é o grafo rotulado, representando o documento XML. Por exemplo, o seguinte documento XML [FLO99]:

```
<person id=1, age=55>
<name>Peter</name>
<address>4711 Fruitdale Ave.</address>
<child>
<person id=3, age=22>
 <name>John</name>
 <address>5361 Columbia Ave.</address>
 <hobby>swimming</hobby>
</person>
</child>
<person>
<person id=2, age=38, child=3>
<name>Mary</name>
<address>4711 Fruitdale Ave.</address>
<hobby>painting</hobby>
</person>
```

Pode ser representado através do grafo da FIGURA 2.

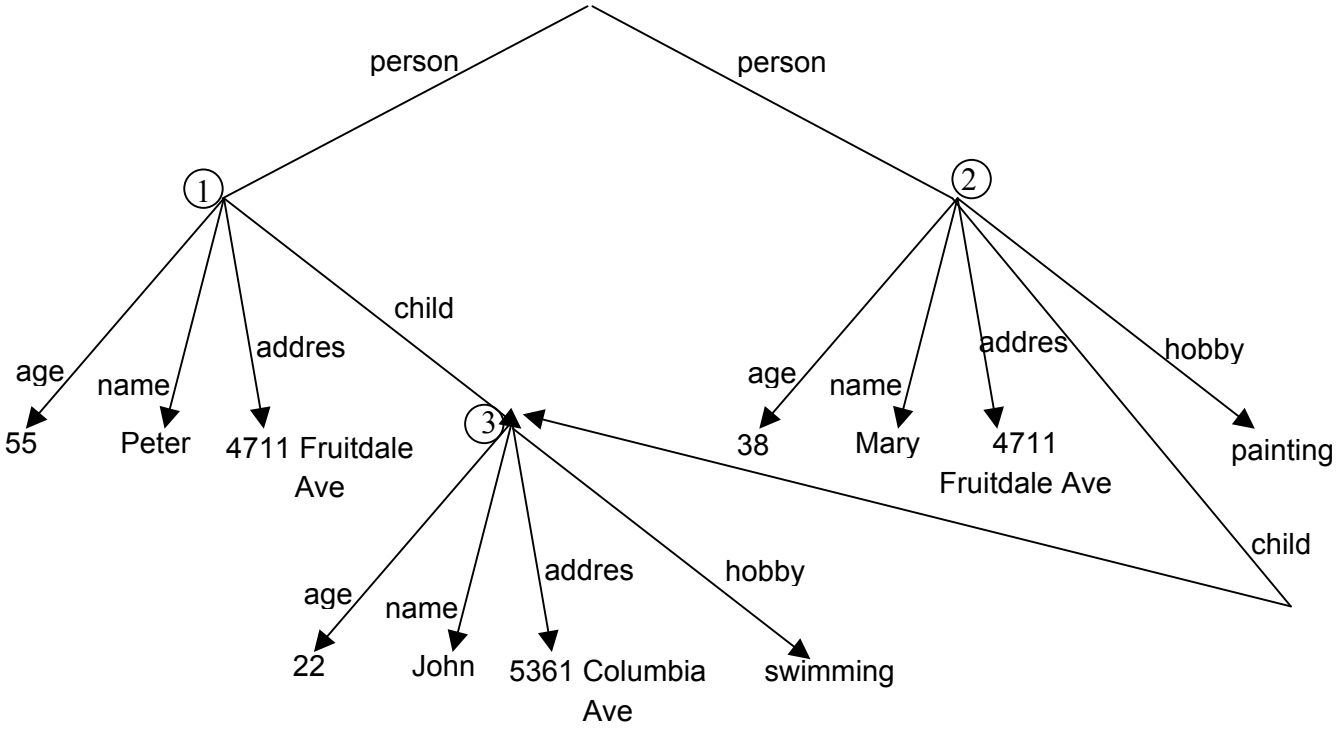

FIGURA 2 - Representação em Grafo do documento XML [FLO99]

O esquema de mapeamento determina quais tabelas são criadas, e em quais tabelas são armazenados os objetos (nós internos do grafo), atributos (arcos) e valores (folhas). As técnicas apresentadas concentram-se no mapeamento de atributos e de valores de dados. São

mostrados quatro mapeamentos para representação de atributos e dois para representação de valores de dados, totalizando oito mapeamentos de esquemas possíveis.

A realização de testes de performance sobre os vários tipos de mapeamentos apresentados em [FLO99], apontaram a combinação de duas técnicas como a melhor combinação em termos de tempo de processamento nas consultas realizadas. Essas técnicas são descritas a seguir.

Para o armazenamento de atributos, a idéia é agrupar todos os atributos com o mesmo nome em uma única tabela. Desse modo, são criadas tantas tabelas *atributo* quantos forem os nomes diferentes de atributos no documento XML. A tabela *atributo* possui a seguinte estrutura:

Atributo (source, ordinal, flag, target)

Onde: source = identificador do objeto fonte

ordinal = ordem do atributo no objeto

flag = indica se o atributo é referência para um objeto ou aponta para um valor

target = identificador do objeto alvo

A chave da tabela é definida por *{source, ordinal}*.

A tabela abaixo mostra a tabela para o atributo *hobby* do exemplo anterior*:*

TABELA 2 - Tabela *Hobby* Resultante do Mapeamento de Atributos e Valores

Source	Ordinal	flag	target
2	5	Valor	Painting
3	4	Valor	Swimming

Para o armazenamento dos valores, a idéia é armazená-los na mesma tabela dos atributos. Essa abordagem, apresentada sob o nome de *inlining,* cria uma coluna para cada tipo de dado (cadeia de caracteres, inteiro, etc).

A tabela abaixo mostra a tabela para o atributo *child* do exemplo anterior*:*

TABELA 3 - Tabela *Child* Resultante do Mapeamento de Atributos e Valores

Source	ordinal	Val_int	Val_string	target
1	4	null	Null	3
2	4	null	null	3

2.2.4 Uso de Ontologias

Do ponto de vista de BD, uma ontologia pode ser vista como uma especificação parcial de um domínio da realidade, que descreve basicamente conceitos, relações entre conceitos e regras de integridade [ERD2001]. Ontologias vêm sendo aplicadas no gerenciamento de dados semi-estruturados como um suporte semântico para o acesso a determinadas informações de interesse presentes em um conjunto de fontes semi-estruturadas. Uma importante vantagem neste contexto é que a ontologia provê uma interpretação semântica unificada para diferentes representações de dados semi-estruturados referentes a um mesmo domínio.

Considerando, por exemplo, um pequeno esquema ontológico para o domínio de artigos científicos, composto pelos conceitos artigo, autor e instituição (FIGURA 3 a)), diversas representações semi-estruturadas (DTDs, por exemplo) podem ser derivadas (FIGURA 3 b), c), d)). Assim, diversos objetos semi-estruturados neste domínio, com diferentes estruturas, podem estar associados a esta mesma ontologia [MEL2000].

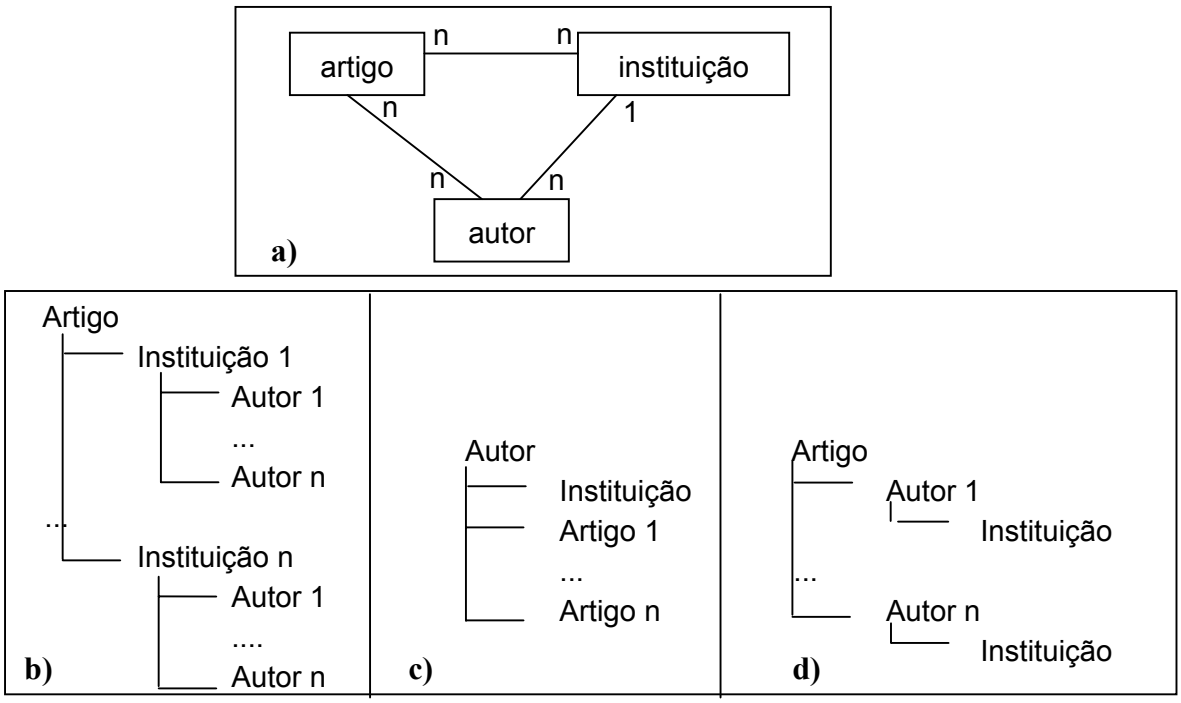

FIGURA 3 - Uma ontologia (a) e algumas representações semi-estruturadas derivadas (b)(c)(d) [MEL2000]

Fornecendo esta base conceitual, conjuntos de documentos XML com estruturas diferentes podem ser acessados, independentes de suas reais representações lineares. Neste sentido, a ontologia também pode ser utilizada para a geração do esquema lógico relacional. A finalidade básica da ontologia é servir como um esquema conceitual. Este esquema conceitual descreve, basicamente, as entidades presentes no domínio de interesse, os relacionamentos

entre essas entidades e suas respectivas cardinalidades. Além disso, a ontologia faz a distinção entre conceitos léxicos e não-léxicos, conforme proposto por [EMB98]:

- Léxicos: objetos que são representáveis diretamente em computador, através de cadeias de bits (números, cadeias de caracteres, etc);
- Não-léxicos: objetos que não têm representação direta em computador (objetos complexos).

Na proposta de [EMB98], um analisador recebe a ontologia e automaticamente gera o esquema relacional. O analisador da ontologia cria um esquema SQL como uma seqüência de comandos *create table*, cujos atributos são objetos da ontologia e cujos tipos são *varchar* para objetos léxicos e *integer* para objetos não léxicos. O analisador também possui um normalizador, o qual produz o esquema em uma forma normal aceitável.

O analisador da ontologia também cria uma lista de objetos, relacionamentos e restrições. Esta lista fornece um mapeamento entre os relacionamentos da ontologia e as declarações das tabelas no esquema SQL, e também fornece as restrições de cardinalidade que indicam quais relacionamentos são um-para-um, um-para-muitos e muitos-para-muitos.

Por exemplo, para a ontologia da FIGURA 4:

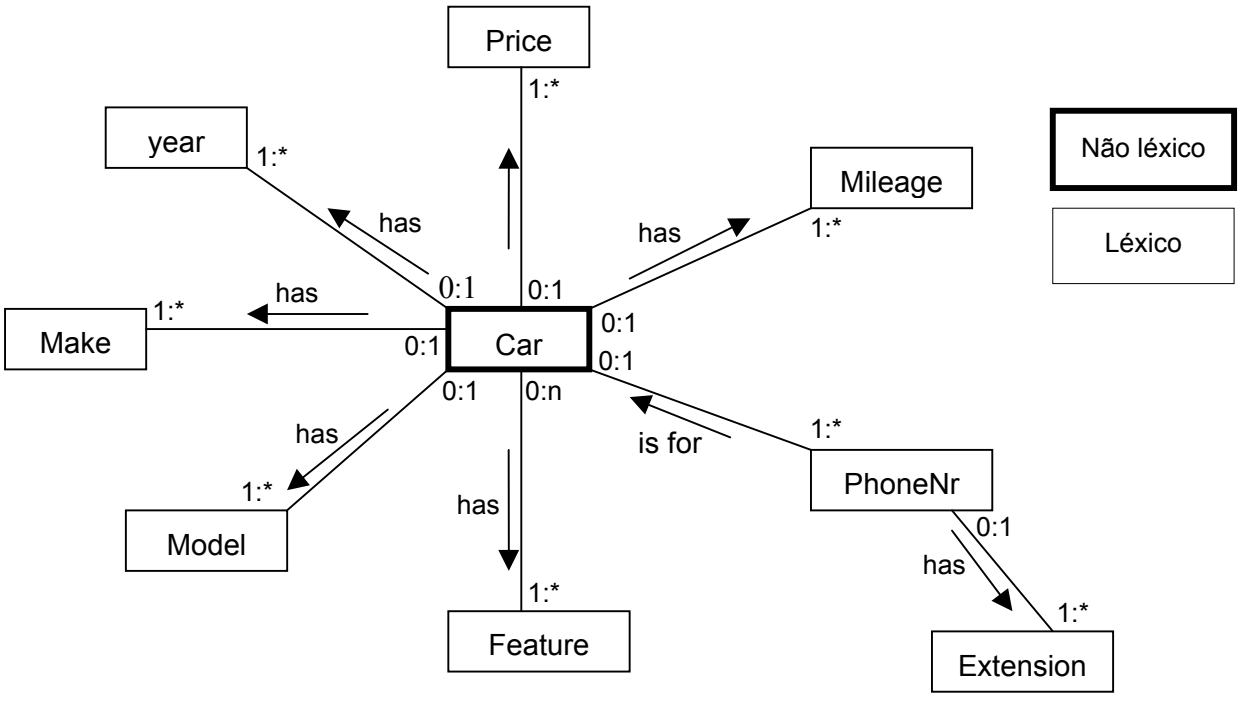

FIGURA 4 - Representação Gráfica da Ontologia - Domínio de Anúncios de Venda de Carros [EMB98]

Tem-se o seguinte esquema lógico gerado:

Car (car, year, make, model, mileage, price, phoneNr)
PhoneNr (phoneNr, extension)
CarFeature (car, feature)

Nesta técnica, a ontologia da aplicação é uma entrada independente, que descreve o domínio de interesse. Quando o domínio é alterado, modifica-se também a ontologia, aplicando-se novamente o processo de geração do esquema lógico relacional.

Outra vantagem a salientar é que consultas que levam em conta a semântica do domínio podem ser formuladas, utilizando-se a ontologia. A ontologia permite que a intenção de consulta do usuário esteja concentrada nos conceitos do domínio e seus relacionamentos e não nas estruturas lógicas de representação de dados semi-estruturados.

Além das facilidades oferecidas a nível de modelagem e consulta, ontologias podem também guiar processos de extração de dados através da manutenção de informações que auxiliem na identificação de atributos de conceitos e relações em representações semi-estruturadas. A definição de um domínio de representação de dados facilita a legibilidade e a precisão de acesso a dados semi-estruturados [MEL 2000].

2.3 Resumo

A técnica *Inferência do esquema lógico a partir da DTD dos documentos* parte do pressuposto que existe uma DTD que valida os documentos XML. Entretanto, documentos XML nem sempre têm uma DTD associada; mesmo quando presente, várias fontes de dados com informação relacionada podem ser validadas contra DTDs diferentes, inviabilizando a geração de um único esquema lógico relacional para a base de dados que materializa estas fontes.

A técnica *Mapeamento específico entre os modelos semi-estruturado e relacional* exige que o mapeamento seja especificado manualmente pelo usuário, utilizando uma linguagem proprietária do sistema. A alternativa do mapeamento automático requer a utilização de algoritmos para a descoberta de padrões entre as fontes de dados.

A técnica *Mapeamento de atributos e valores* tem como ponto de partida um grafo rotulado, o qual representa um único documento XML. Novamente, o esquema gerado não se adequa a outros documentos XML que apresentem estrutura diferente do documento utilizado como base para a geração do esquema relacional.

Embora funcionem bem para determinadas aplicações, as propostas acima falham em um ponto importante. Os algoritmos de transformação capturam somente a estrutura de uma DTD ou de um documento XML específico, e ignoram o conhecimento semântico embutido nas fontes de dados. Tentando suprir esta deficiência, o *Uso de ontologias* apresenta-se como uma

técnica bastante flexível. Vários documentos XML, com DTDs diferentes ou mesmo ausentes, podem ser descritos por uma mesma ontologia. A ontologia fornece um vocabulário compartilhado que integra as diferentes fontes XML, torna a informação acessível de uma maneira uniforme, e faz a mediação entre os termos conceituais utilizados pelo usuário que acessa a informação e as marcações utilizadas nos documentos XML. Desse modo, a geração do esquema lógico relacional baseado em uma ontologia aparece como uma boa alternativa.

A técnica proposta neste trabalho utiliza uma ontologia para a geração do esquema lógico da base de dados relacional, conforme apresentado na seção 4.3.

3 Identificação de Instâncias

Nesta seção é apresentado o problema da identificação de instâncias e as principais abordagens encontradas na literatura para a sua solução.

3.1 Visão Geral

O esquema lógico gerado para o banco de dados relacional, apresentado na seção 4.3, especifica um campo identificador para cada instância XML materializada: *id_autor, id_artigo*, e assim por diante. Em algumas tabelas geradas, este campo é a própria chave primária da tabela. A chave primária fornece um valor único, através da qual pode-se acessar um determinado registro de uma tabela. No entanto, dados XML como propostos originalmente, não carregam necessariamente a noção de chave primária ou identificador de objeto.

O conceito que mais assemelha-se a um identificador de objeto em XML é a criação de um atributo do tipo ID para cada elemento. Este atributo fornece acesso único ao elemento dentro de um documento XML. O documento a seguir utiliza um atributo *id* do tipo ID para identificação de seus elementos *autor*:

```
... <artigo>Caching XML Documents</artigo>
      <autor id=101>
           <nome>Marcos Santos</nome>
          <email>santos@inf.ufrgs.br</email>
       </autor>
       <autor id=102>
           <nome>Joao Pereira</nome>
          <email>pereira@inf.ufrgs.br</email>
       </autor>...
  <artigo>
```

Entretanto, a criação deste atributo é opcional. Mesmo quando presente, este atributo restringe-se a distinguir as instâncias de um determinado tipo dentro de um documento XML, não sendo aplicáveis à integração de diferentes documentos. Desse modo, diferentes instâncias XML vindas de diferentes fontes de dados podem estar modelando o mesmo objeto do mundo real.

Por exemplo, as instâncias de dados “Marcos Alberto Santos” e “Marcos A. Santos”, vindas de duas fontes diferentes de dados, podem se referir ao mesmo objeto *autor* do mundo real. O processamento de consultas ou a construção de visões sobre estas fontes exige a *integração das instâncias*. Neste contexto, integrá-las significa identificar quais instâncias de dados representam o mesmo objeto do mundo real, bem como resolver ambigüidades de representação deste objeto. Desse modo, é necessária a adoção de um mecanismo que faça a identificação das instâncias na integração dos dados XML. O estado da arte na identificação de instâncias é apresentado nas seções 3.2 e 3.4.

Além disso, após duas instâncias serem identificadas como representando um mesmo objeto do mundo real, os valores de suas propriedades podem apresentar valores conflitantes. No contexto deste trabalho, considera-se que propriedades de objetos são representadas em XML por elementos atômicos e atributos, ou seja, pelas folhas da árvore representativa de uma instância XML. Um exemplo deste tipo de conflito é a presença de dois endereços diferentes para um mesmo autor, conforme mostrado abaixo:

```
    Fonte de Dados 1:
... <autor>
          <nome>Marcos Santos</nome>
          <endereco>Rua x, 110</endereco>
    </autor>...

    Fonte de Dados 2:
... <autor>
          <nome>Marcos Santos</nome>
          <endereco>Rua y, 220</endereco>
    </autor>...
```

Considerando que ambas fontes modelam o mesmo objeto *autor* do mundo real, a propriedade *endereço* apresenta valores conflitantes. Para o tratamento deste tipo de conflito, várias abordagens são propostas na literatura para bases de dados estruturadas. Estas técnicas são apresentadas na seção 3.3.

3.2 Identificação de Instâncias em Bancos de Dados Heterogêneos

Esta seção apresenta as principais técnicas utilizadas na área de Bancos de Dados Heterogêneos para o tratamento de identificação de instâncias.

3.2.1 Chave Universal

Este é o método mais simples e direto de integração de dados. Baseia-se na existência de uma chave comum entre as instâncias a serem integradas [LIM94, WIE97, PAP96, MOT81, MAN2000, ALB96]. Desse modo, duas instâncias de dados são consideradas o mesmo objeto do mundo real sempre que possuírem o mesmo valor para a chave primária em comum.

Esta técnica é bastante utilizada na integração de bancos de dados relacionais, quando existe uma chave em comum entre as fontes de dados. Por exemplo, considerando que cada uma das tabelas a seguir venha de uma fonte de dados diferente:

TABELA 4 - Tabela Artigo1 a ser Integrada pelo Método da Chave Universal

Id_artigo	**Titulo**
1	Caching XML Views
2	Semistructured Data
3	Database Integration

TABELA 5 - Tabela Artigo2 a ser Integrada pelo Método da Chave Universal

Id_artigo	**Ano_publicacao**	**Evento**
1	1999	SBBD
2	2000	CLEI
3	1999	SBBD

Considerando que *id_artigo* é chave primária em ambas tabelas, a integração das fontes de dados dá-se pela chave em comum.

Supondo que todos os atributos presentes nas duas fontes de dados façam parte do sistema integrado, a integração das instâncias realiza-se por *id_artigo:*

TABELA 6 - Tabela Artigo Resultante da Integração pelo Método da Chave Universal

Id_artigo	**Titulo**	**Ano_publicacao**	**Evento**
1	Caching XML Views	1999	SBBD
2	Semistructured Data	2000	CLEI
3	Database Integration	1999	SBBD

3.2.2 Equivalência de Chaves Especificada pelo Usuário

Nesta solução, a idéia é introduzir um sistema independente de identificadores únicos globais, os quais devem ser mapeados para identificadores locais de cada fonte de dados participante. Esta abordagem requer que o usuário especifique equivalência entre as instâncias, usando por exemplo, uma tabela de mapeamento dos identificadores locais de cada fonte para os identificadores globais do sistema integrado. Parte-se do pressuposto que as fontes de dados têm como identificar suas instâncias. Esta técnica é apresentada em [AHM91, RED94, SHO93].

O sistema *Rufus* [SHO93] utiliza esta abordagem. Dados semi-estruturados são armazenados como instâncias de classes orientadas a objetos. Estas classes descrevem tipos de arquivos do usuário, por exemplo: mensagens de *mail*, código C de uma determinada aplicação, arquivos de imagem, etc. O sistema inclui um classificador, o qual automaticamente determina a classe *Rufus* à qual o arquivo de usuário pertence. Na importação dos dados originais para as classes do sistema, um identificador de objeto é criado sempre que o arquivo original dos dados esteja sendo importado pela primeira vez; se o arquivo é modificado e re-importado, seu identificador permanece o mesmo. Para isso, supõe-se que cada arquivo original nas fontes de dados tem como ser identificado. Quando o arquivo é importado, seu identificador é descoberto pelo método construtor da classe *Rufus*. A seguir, é realizado um mapeamento persistente deste identificador para o identificador de objeto utilizado nas classes do sistema. Se o identificador da fonte de dados já é conhecido pelo sistema, então o identificador do objeto é reusado. A estratégia deste sistema funciona bem para fontes de dados que tem um identificador único intrínseco.

Um mapeamento manual também pode ser realizado entre os identificadores das fontes de dados e os identificadores no sistema integrado. Em [AHM91], uma tabela construída pelo usuário pode ser utilizada na especificação de equivalência de identificadores. Neste caso, a desvantagem é que a tabela de mapeamento pode ser extremamente grande e de difícil manutenção, manuseada pelo próprio administrador do banco de dados, de uma maneira não automatizada.

3.2.3 Equivalência de Atributos

Não havendo uma chave comum entre as instâncias a integrar, algumas propostas sugerem o uso de atributos comuns. Todos os atributos em comum entre as fontes de dados ou apenas alguns deles (especificados pelo administrador do sistema integrado) podem ser utilizados para determinar a equivalência de objetos [ZHO95, CHA91, LIM98, WAN89]. Por exemplo, supondo que o atributo *nome_departamento* seja comum a duas fontes de dados a serem integradas; neste caso, o sistema pode basear-se neste atributo para realizar a correspondência entre instâncias de dados do tipo *departamento*.

Quando mais de um atributo é utilizado para realizar a correspondência de instâncias de dados, pode ser gerado um valor resultante desta comparação. Este valor de comparação mede o grau de similaridade entre as duas instâncias: para cada par de registros de duas fontes de dados, é processado um valor de comparação. Inicialmente, é gerado um vetor. Este vetor armazena o resultado da comparação entre os atributos que estão sendo comparados para as instâncias. A posição do vetor *(i)* refere-se ao atributo e o valor armazenado nesta posição é definido como:

Vetor(i) = 1 se o atributo apresenta o mesmo valor para as duas instâncias de dados

Vetor(i) = 0 se o atributo apresenta valor diferente para as duas instâncias de dados

[CHA91] apresenta esta abordagem. Por exemplo, considerando duas relações *empregado* e *funcionário,* com os seguintes atributos e valores de atributos:

TABELA 7 - Tabela Empregado a ser Integrada pelo Método da Equivalência de Atributos

nome	endereço	data_nascimento	departamento
João Souza	Rua x, 110	01/10/1970	Recursos Humanos
Marcos Santos	Rua y, 220	12/12/1965	Produção

TABELA 8 - Tabela Funcionário a ser Integrada pelo Método da Equivalência de Atributos

nome	Departamento	salario	data_nascimento
João Carlos Souza	Recursos Humanos	3200	01/10/1970
Marcos Santos	Produção	2700	12/12/1965

Supondo que os atributos *nome, data_nascimento* e *departamento* sejam utilizados para realizar a correspondência de instâncias; para cada par de registros das fontes de dados, é gerado o vetor resultante da comparação do valor dos atributos *nome, data_nascimento* e *departamento:*

a) Vetor resultante da comparação do 1º registro de *empregado* com o 1º registro de *funcionario:*

Vetor = [0 1 1]

b) Vetor resultante da comparação do 1º registro de *empregado* com o 2º registro de *funcionario:*

Vetor = [0 0 0]

c) Vetor resultante da comparação do 2º registro de *empregado* com o 1º registro de *funcionario:*

Vetor = [0 0 0]

d) Vetor resultante da comparação do 2º registro de *empregado* com o 2º registro de *funcionario:*

Vetor = [1 1 1]

O valor de comparação pode agora ser definido como uma função destes vetores. Esta função é implementada à escolha do usuário. Um modelo probabilístico pode ser utilizado na geração deste valor de comparação [CHA91]. Este modelo probabilístico deve levar em consideração que a probabilidade de duas instâncias de dados serem a mesma entidade do mundo real é tanto maior quanto for o número de posições ocupadas com o valor *1* nos vetores acima. Por exemplo, no caso anterior, poder-se-ia considerar que o vetor a) representa duas instâncias de dados de um mesmo objeto do mundo real (dos três atributos comparados, dois deles apresentam o mesmo valor); do mesmo modo, poder-se-ia considerar que o vetor d) representa duas instâncias de dados de um mesmo objeto do mundo real (dos três atributos comparados, todos apresentam o mesmo valor);

3.2.4 Consultas Definidas pelo Usuário

Uma consulta pode ser definida a cada tipo de objeto do mundo real, a fim de construir uma propriedade única identificadora das instâncias deste tipo. Sempre que duas instâncias de um mesmo tipo retornarem o mesmo resultado para a consulta, estas serão consideradas um mesmo objeto do mundo real.

Esta técnica objetiva identificar instâncias através de propriedades naturais, intrínsecas do objeto. [GOG95] enfatiza a importância desta abordagem, uma vez que não utiliza mecanismos artificiais de identificação, como por exemplo, valores numéricos sem significado semântico gerados automaticamente na inserção dos registros nas fontes de dados.

Por exemplo, considerando que uma instância do tipo *cidade* seja identificada pelo seu nome e a UF a qual esta pertence. A consulta definida sobre este tipo de objeto retorna o valor destas propriedades para as instâncias deste tipo, e considera-as o mesmo objeto do mundo real sempre que os resultados da consulta forem os mesmos. Supondo as seguintes relações:

TABELA 9 - Tabela Endereço a ser Integrada pelo Método Consultas Definidas pelo Usuário

Rua	Nome_cidade	UF	...
Andradas	Santa Maria	RS	
Independencia	Porto Alegre	RS	
Salgado Filho	Santa Maria	RS	

TABELA 10 - Tabela Empregado a ser Integrada pelo Método Consultas Definidas pelo Usuário

Cod_empr	Cidade	estado	...
1	Santa Maria	RS	
2	São Paulo	SP	
3	Santa Maria	RS	

Para a identificação das instâncias de dados do tipo *cidade,* são submetidas as consultas abaixo:

a) Select nome_cidade, UF
From ENDERECO

Esta consulta retorna:

TABELA 11 - Tabela Temp1 Resultante da Consulta Submetida à Tabela Endereço

Nome_cidade	UF
Santa Maria	RS
Porto Alegre	RS

b) Select cidade, estado
From EMPREGADO

Esta consulta retorna:

TABELA 12 - Tabela Temp2 Resultante da Consulta Submetida à Tabela Empregado

cidade	estado
São Paulo	SP
Santa Maria	RS

Analisando as tabelas Temp1 e Temp2, nota-se que existem dois registros com os mesmos valores para as propriedades que modelam o nome e a UF da cidade (*Santa Maria, RS*). O primeiro registro de Temp1 e o segundo registro de Temp2 representam a mesma cidade . Desse modo, estas duas instâncias referem-se à mesma entidade do mundo real.

3.2.5 Regras de Inferência

Esta técnica baseia-se em regras de inferência para a derivação de identificadores. A proposta utiliza informação semântica extra para automatizar o processo de identificação entre relações que não compartilham uma chave em comum. Para isso, propõe o uso de uma chave estendida e regras de identidade (equivalência entre chaves estendidas) para identificar instâncias. Esta técnica não exige uma chave comum entre as fontes, mas parte do pressuposto que cada fonte tem como identificar suas instâncias através de uma chave primária. Desse modo, basicamente, infere conhecimento extra nas outras fontes para deduzir o valor da chave estendida.

A chave estendida pode ser definida como o conjunto mínimo de atributos necessários para identificar unicamente uma instância do mundo real [LIM93]. Geralmente, a chave estendida é a união das chaves das fontes de dados participantes, e pode ser considerada o identificador destas instâncias no sistema integrado. Como esta técnica não exige que todas as fontes de dados a serem integradas compartilhem uma chave em comum, o valor da chave estendida deve ser inferido nas fontes de dados.

Considerando as seguintes relações que modelam um conjunto de entidades *E* do tipo *restaurante*:

TABELA 13 - Tabela R a ser Integrada pelo Método Regras de Inferência

nome	**cozinha**	**rua**
Giorgio	italiana	24 de outubro
Giorgio	chinesa	Protásio Alves

TABELA 14 - Tabela S a ser Integrada pelo Método Regras de Inferência

nome	**especialidade**	**cidade**
Giorgio	pizza	Porto Alegre

Onde: {nome, cozinha} é chave primária de R e {nome} é chave primária de S.

A chave estendida definida sobre estas relações pode ser descrita pela união das chaves das fontes de dados participantes *R* e *S:*

$$\{R.nome, R.cozinha, S.nome\}$$

A partir da definição da chave estendida, pode-se derivar uma regra de equivalência de chave estendida, a qual especifica o critério a ser satisfeito para que duas instâncias de dados sejam consideradas a mesma entidade do mundo real:

$$\forall e1, e2 \in E, (e1.nome = e2.nome) \wedge (e1.cozinha = e2.cozinha) \rightarrow (e1 \equiv e2)$$

Sempre que duas instâncias do tipo *restaurante* tiverem o mesmo nome e o mesmo tipo de cozinha, considerar-se-á que modelam um mesmo objeto do mundo real.

No entanto, uma vez que a relação *S* não possui o atributo *cozinha*, esta regra não pode ser diretamente aplicada. Desse modo, faz-se necessário estender a relação *S*, incluindo a informação necessária para o atributo *cozinha.* Este conhecimento extra pode ser adquirido através do projetista do sistema, algoritmos de descoberta de conhecimento, etc [WAN89].

Por exemplo, supondo que todo o restaurante especializado em *pizza* é de cozinha *italiana*. A informação extra necessária para derivar o valor deste atributo pode ser expressa como:

I1: (E.especialidade=”pizza”) $\rightarrow$ *(E.cozinha=”italiana”).* Assim, a relação *S* pode ser estendida em *S’* para incluir o valor do atributo *cozinha*:

TABELA 15 - Tabela S' Estendida a partir da TABELA 14

Nome	**especialidade**	**cidade**	**cozinha**
Giorgio	pizza	Porto Alegre	italiana

Conforme mostrado anteriormente, o critério a ser satisfeito neste exemplo para que duas instâncias de dados sejam consideradas a mesma entidade do mundo real é:

$\forall e1, e2 \in E, (e1.nome = e2.nome) \wedge (e1.cozinha = e2.cozinha) \rightarrow (e1 \equiv e2)$

Analisando as relações apresentadas, tem-se que a primeira tupla da relação *R* tem correspondência com a tupla da relação *S’* estendida. Desse modo, conclui-se que estas duas tuplas referem-se à mesma entidade do mundo real.

3.2.6 Funções Booleanas Definidas pelo Projetista

Quando não há uma chave comum entre os dados a serem integrados e quando não existe nenhum atributo em comum que possa ser utilizado na identificação das instâncias, o projetista pode definir uma função para tal propósito. A função recebe parâmetros de entrada, e retorna um valor booleano após o processamento destes valores. O tipo deste processamento é definido pelo administrador do sistema.

Um exemplo desta técnica é apresentado em [ZHO95], o qual utiliza uma função *close_names()*; esta função retorna um valor booleano, dependendo da similaridade entre duas cadeias de caracteres passadas como argumentos. Esta função pode, por exemplo, ignorar abreviaturas de nomes no processo de identificação de instâncias, e considerar que “Marcos Santos” e “Marcos A. Santos” referem-se ao mesmo objeto do mundo real.

3.3 Resolução de Conflitos nas Propriedades de Objetos em Bancos de Dados Heterogêneos

Dadas duas instâncias que representam a mesma entidade do mundo real, as diferenças nos valores de suas propriedades equivalentes são conhecidas como conflitos. A seguir, são apresentadas algumas técnicas encontradas na literatura para o tratamento deste tipo de conflito.

3.3.1 Funções de Agregação

Nesta abordagem, são utilizadas as funções de agregação das linguagens de consulta (mínimo, máximo, média, etc) para resolver conflitos de valores de atributos. A partir de um conjunto fixo de funções de agregação fornecido, um único valor é considerado para o atributo no sistema integrado. Exemplificando, caso o valor do preço de um livro esteja representado com valores diferentes em duas fontes de dados, poder-se-ia utilizar uma função de mínimo e considerar apenas o menor valor para a propriedade preço no sistema integrado.

Por exemplo, considerando as seguintes relações:

TABELA 16 - Tabela R a ser Integrada Utilizando Funções de Agregação

Id_livro	Nome_livro	Preco
100	Introducao a XML	43,69
102	Bancos de Dados	53,00

TABELA 17 - Tabela S a ser Integrada Utilizando Funções de Agregação

Id_livro	Editora	Valor
100	Makron Books	45,90

Supondo que as instâncias do tipo *livro* são identificadas por *id_livro*, tem-se que o primeiro registro da relação R e o registro da relação S referem-se ao mesmo objeto do mundo real. Entretanto, as duas instâncias apresentam valores conflitantes para a propriedade *preço* do livro. Neste caso, poder-se-ia, por exemplo, aplicar uma função de agregação de mínimo; o valor resultante da aplicação desta função é o valor a ser considerado para a propriedade *preço* no sistema integrado. Para o livro identificado por *id=100*, o valor da propriedade no sistema integrado será o menor valor presente em todas as relações participantes, ou seja, R$ 43,69.

Esta técnica é proposta em [DAY83, RED94, ALB96, LIM98]. A desvantagem desta técnica é que as funções utilizadas podem restringir o tipo de atributo ao qual podem ser aplicadas; como exemplo, função de mínimo só pode ser aplicada a valores numéricos.

3.3.2 Valores Parciais

Quando diferentes valores de uma propriedade não podem ser mapeados para um único valor definido no sistema integrado, pode-se produzir um valor parcial. Este valor parcial pode ser definido como um conjunto de valores possíveis para esta propriedade. A idéia é utilizar um conjunto de operadores relacionais estendidos; estes operadores formalizam operações sobre os valores parciais, manipulando esta informação.

Esta técnica, basicamente, realiza [DeM89]:

- valores reais dos atributos são mapeados para atributos virtuais. Os valores que resultam deste mapeamento podem ser parciais ou totais;
- operadores estendidos são aplicados sobre as relações resultantes do passo anterior.

Esta proposta é apresentada em [DeM89]. Por exemplo, considerando as seguintes relações [DeM89]:

TABELA 18 - Tabela A a ser Integrada Utilizando Valores Parciais

Nome_restaurante	Tipo_cozinha
Brandy Ho	Hunan
Hunan	Hunan
Mandarin	Mandarin
China West	Cantonese

TABELA 19 - Tabela B a ser Integrada Utilizando Valores Parciais

Nome_restaurante	Localizacao	Tipo_cozinha
Brandy Ho	North Beach	Chinese
Empress of China	Chinatown	Chinese
Five Happiness	Richmond	Chinese
China West	Richmond	Chinese
Asia Garden	Chinatown	Chinese

Considera-se que as instâncias de dados são integradas pelo atributo *nome_restaurante*. Supondo que a consulta "selecione nomes de restaurantes com comida *hunan* em *North Beach*". O atributo *tipo_cozinha* apresenta valores conflitantes nas duas fontes de dados.

Sabendo-se que todo restaurante chinês é especializado em *cantonese, hunan* ou *mandarin*, por exemplo. Desse modo, é realizado um mapeamento dos valores deste atributo na relação B para os valores deste atributo na relação A. Este mapeamento resulta em valores parciais ([*cantonese, hunan, mandarin*]) na relação do sistema integrado. Realizando a operação de união entre as duas relações, baseada no atributo *nome_restaurante* para a integração das instâncias, tem-se:

TABELA 20 - Tabela Resultante da Integração Utilizando Valores Parciais

Nome_restaurante	Localizacao	Tipo_cozinha
Brandy Ho	North Beach	Hunan
Empress of China	Chinatown	[cantonese, hunan, mandarin]
Five Happiness	Richmond	[cantonese, hunan, mandarin]
China West	Richmond	Cantonese
Asia Garden	Chinatown	[cantonese, hunan, mandarin]
Hunan	-	Hunan
Mandarin	-	Mandarin

Operações de seleção estendidas são aplicadas nos dados do sistema integrado para a manipulação dos valores parciais. A presença de valores parciais no atributo *tipo_cozinha* e a ausência de valor no atributo *localizacao* produz resultados parciais para a consulta submetida "selecione nomes de restaurantes com comida *hunan* em *North Beach*". Resultados parciais são expressos em tuplas cujo valor para o atributo *status* seja definido como *maybe*. Desse modo, a relação a seguir é retornada para a consulta submetida:

TABELA 21 - Tabela Resultante da Consulta Submetida à TABELA 20

Nome_restaurante	Status
Brandy Ho	True
Hunan	Maybe

3.3.3 Valores Parciais Probabilísticos

Esta proposta atribui probabilidades aos valores conflitantes dos atributos. Operações estendidas de seleção e junção filtram as tuplas que não satisfazem a condição da consulta com a probabilidade desejada. Por exemplo, considerando o caso anterior, pode-se aplicar algum método de atribuição de probabilidades aos valores conflitantes da propriedade

endereço: endereço 1 = probabilidade 0,66; endereço 2 = probabilidade 0.33. Uma consulta especificando uma probabilidade mínima desejada de 0,5 retornaria apenas o valor do endereço 1 para o usuário.

Cada fonte de dados atribui probabilidades aos valores possíveis para um determinado atributo. A soma das probabilidades dos vários valores para um atributo em uma fonte de dados é *um*. Desse modo, basicamente a probabilidade de um valor de atributo no sistema integrado é definida como o produto das probabilidades para este valor de atributo nas fontes de dados. Esta técnica é apresentada em [LIM94].

Por exemplo, considerando duas fontes de dados do domínio de restaurantes; θespecialidade é o conjunto de todos as especialidades possíveis oferecidas por um restaurante:

θespecialidade = {american, hunan, sichuan, cantonese, mughalai, italian}

Considerando um determinado restaurante chinês, cuja especialidade não é completamente determinada. Baseado, por exemplo, em um modelo de votação, poder-se-ia ter as seguintes assertivas sobre este restaurante em uma fonte de dados 1 [LIM94]:

a) $m_1(\{cantonese\}) = ½$

b) $m_1(\{hunan, sichuan\}) = 1/3$

c) $m_1(\theta especialidade) = 1/6$

As três definições acima demostram que nesta fonte de dados:

a) a metade dos pratos é de especialidade *cantonese;*

b) um terço dos pratos não pode ser classificada como *hunan* puro ou *sichuan* puro;

c) um sexto dos pratos não dispõe de informação disponível sobre a especialidade.

Uma outra fonte de dados 2 apresenta as seguintes probabilidades para o atributo especialidade:

a) $m_2(\{cantonese, hunan\}) = ½$

b) $m_2(\{hunan\}) = 1/4$

c) $m_2(\theta especialidade) = 1/4$

As três definições acima demostram que nesta fonte de dados:

a) a metade dos pratos não pode ser classificada como *cantonese* puro ou *hunan* puro;

b) um quarto dos pratos é de especialidade *hunan* puro;

c) um quarto dos pratos não dispõe de informação disponível sobre a especialidade.

Baseado nestas definições, pode ser gerada uma tabela com as probabilidades para os valores do atributo no sistema integrado [LIM94]:

TABELA 22 - Tabela de Probabilidades para os Valores do Atributo *Especialidade*

	m_2({cantonese,hunan})= ½	**m_2({hunan})=1/4**	**m_2(θespecialidade)=1/4**
m_1({cantonese})=½	{ca} ¼	ϕ 1/8	{ca} 1/8
m_1({hunan, sichuan})=1/3	{hu} 1/6	{hu} 1/12	{hu, si} 1/12
m_1(θespecialidade) = 1/6	{ca,hu} 1/12	{hu} 1/24	θ 1/24

Cada entrada interna desta tabela é a interseção de um par de membros. A probabilidade da entrada interna é definida como o produto das probabilidades dos dois membros. O valor nulo ϕ ocorre porque {*hunan}* e {*cantonese*} não tem elemento em comum.

A probabilidade do restaurante ser de uma determinada especialidade é definida como o somatório das probabilidades das entradas internas da tabela para esta especialidade, dividido por *(1-k)*, onde:

$$k = \sum_{x \cap y = \phi} m_1(x).m_2(y)$$

Neste exemplo, k = 1/8.

Analisando a tabela, tem-se:

- a probabilidade do restaurante ser de especialidade *cantonese* puro é definida por (1/4 + 1/8) / (1 – 1/8) = 3/7 ≈ 42.85%;
- a probabilidade do restaurante ser de especialidade *hunan* puro é definida por (1/6 + 1/12 + 1/24) / (1 – 1/8) = 1/3 ≈ 33.33%, e assim por diante.

Consultas do usuário especificam a probabilidade desejada. [LIM94] propõe operações estendidas de seleção e junção para a filtragem das tuplas retornadas para a consulta.

3.3.4 Armazenamento de Todos os Valores Conflitantes dos Atributos

Esta técnica propõe armazenar todos os valores conflitantes dos atributos encontrados nas fontes de dados. Para isso, o modelo de dados do sistema integrado é estendido, a fim de

dar suporte às consultas do usuário sobre atributos com os valores originais das fontes e com os valores resolvidos no sistema integrado. Os valores resolvidos podem ser derivados utilizando-se, por exemplo, uma função de agregação (apresentada na seção 3.3.1).

Diferentemente da proposta *valores parciais probabilísticos*, este técnica não atribui probabilidades aos valores conflitantes dos atributos, mas sim possibilita que todos os valores encontrados nas fontes para uma mesma propriedade de um objeto do mundo real sejam consultados, assim como também o valor resolvido no sistema integrado. Desse modo, o sistema integrado armazena:

- os valores conflitantes do atributo;
- o valor resolvido deste atributo no sistema integrado, quando possível. Na inexistência de uma função de resolução para o conflito, o valor resolvido do atributo no sistema integrado é representado como nulo.

O valor original de um atributo é descrito por um conjunto de pares (*valor, DB_id*), onde *valor* denota o valor do atributo encontrado na fonte identificada por *DB_id*.

Por exemplo, considerando as seguintes relações [LIM98]:

TABELA 23 - Tabela A a ser Integrada Utilizando Armazenamento de todos os Valores Conflitantes de Atributos

Nome_empregado	Posicao	Salario
John	Estagiario	1000
Kim	Secretario	1500
Chen	Engenheiro	2500

TABELA 24 - Tabela B a ser Integrada Utilizando Armazenamento de todos os Valores Conflitantes de Atributos

Nome_empregado	Posicao	Salario
Kim	Assistente	1500
Chen	Engenheiro	2600
John	Estagiario	1000

Considerando que as instâncias das duas relações sejam identificadas pelo atributo *nome_empregado*. Baseado nesta definição, o salário do empregado *Chen* apresenta valores conflitantes nas duas fontes de dados. Supondo que existe uma função para a resolução deste conflito, definida como a média aritmética de todos os valores conflitantes deste atributo:

Salario_resolvido = (A.salario + B.salario)/2 = 2550

Da mesma maneira, o atributo *posicao* apresenta valores conflitantes para o empregado *Kim.* Supondo que não exista uma função para a resolução deste conflito; assim, o valor deste atributo para o empregado *Kim* no sistema integrado é definido como nulo.

A relação a seguir mostra os valores originais e resolvidos para os atributos *nome_empregado, posicao* e *salario* no sistema integrado [LIM98]:

TABELA 25 - Tabela Resultante da Integração Utilizando Armazenamento de todos os Valores Conflitantes de Atributos

Tipo_valor	oid	Nome_empregado	Posicao	Salario
Valor_original	E1	(John, A) (John, B)	(estagiario,A)(estagiario,B)	(1000,A)(1000,B)
Valor_resolvido		John	estagiario	1000
Valor_original	E2	(Kim, A) (Kim, B)	(secretario,A)(assistente,B)	(1500,A)(1500,B)
Valor_resolvido		Kim	nulo	1500
Valor_original	E3	(Chen, A) (Chen, B)	(engenheiro,A)(engenheiro, B)	(2500,A)(2600,B)
Valor_resolvido		Chen	engenheiro	2550

Assim, o usuário pode consultar todos os valores conflitantes para o atributo, assim como também o valor resolvido no sistema integrado, quando existente.

Esta técnica é utilizada em [LIM98]. Um modelo de objetos é apresentado, assim como uma linguagem de consulta estendida. O modelo apresentado dá suporte ao armazenamento dos valores originais e resolvidos dos atributos.

3.4 Identificação de Instâncias XML

Há duas maneiras para identificar um objeto em XML. Primeiro, pode-se utilizar posições para descrever um caminho dentro de um documento. Nesta técnica, um elemento XML é identificado pela sua posição inicial na fonte de origem. Por exemplo, o terceiro artigo na segunda conferência [MAG2001]. Entretanto, posições podem mudar durante as atualizações nas fontes de dados (como a inserção de um novo *autor*).

A segunda técnica é adotar um modelo de dados semi-estruturado não ordenado. Nós são identificados por identificadores de objetos. Nesta técnica, as propostas encontradas na literatura seguem duas abordagens: assumem a existência do identificador do objeto ou fazem a inserção deste identificador automaticamente.

-assumem a existência do identificador de objeto : esta abordagem utiliza modelos de dados que assumem a existência de um identificador de objeto à cada elemento XML a ser identificado [FLO99**,** LIE99, FLO99a, DEU99, ABI98]. Neste caso, a integração das instâncias pode ser realizada pela comparação dos valores de seus identificadores;

-inserem identificadores de objetos: na ausência de um identificador, estas abordagens realizam automaticamente sua inserção nos elementos XML a serem identificados [LIE99, DEU99, FLO99a]. A criação do identificador do objeto pode levar em conta o caminho percorrido desde a raiz do documento XML até o elemento a ser identificado, o conteúdo do elemento, gerados aleatoriamente, entre outras abordagens. Neste caso, a integração das instâncias nem sempre pode basear-se no identificador de objeto. Por exemplo, no caso em que foi gerado aleatoriamente, o valor do identificador deste objeto não tem relevância na integração dos dados.

3.5 Resumo

Identificação de objetos em sistemas de bancos de dados únicos é relativamente simples. Muitos SGBDs orientados a objetos têm desenvolvido algumas abordagens para tratar esta questão. Entretanto, identificação de objetos em um sistema de gerenciamento *multidatabase* heterogêneo é difícil, porque logicamente diferentes objetos podem ter o mesmo identificador em diferentes fontes de dados. A solução usual para a colisão de identificadores de objetos em múltiplas fontes de dados é introduzir um sistema independente de identificadores únicos globais, os quais devem ser mapeados para identificadores locais de cada banco de dados local participante.

As propostas encontradas na literatura para integração de dados XML geralmente partem do pressuposto que os elementos já possuem identificadores. Em outras palavras, baseiam-se na existência de uma chave comum entre as instâncias a serem integradas. Entretanto, dados XML não carregam necessariamente a noção de identificador de objeto. Deste modo, quando estes identificadores estão ausentes, sistemas proprietários fazem a sua inclusão. Dependendo do método utilizado para isso, os identificadores podem não ter relevância e não serem úteis para a integração das instâncias.

A proposta deste trabalho baseia-se na abordagem OWA (*Open World Assumption*), pois deseja-se integrar dados da WEB. Nesta abordagem, não se pode esperar que os dados possuam identificadores. Com isto, também estão excluídas as técnicas que baseiam-se na existência de um identificador na fonte de dados (*Equivalência de chave especificada pelo usuário*, *Regras de inferência*).

Algumas outras técnicas utilizadas na integração de dados em bancos de dados heterogêneos baseiam-se na existência de um identificador na fonte de dados, ainda que este não seja comum a todas as fontes. Desse modo, uma tabela de equivalência de chaves,

especificada pelo usuário, pode ser definida. Como dados XML não carregam necessariamente um identificador, essas técnicas também não são viáveis de serem utilizadas.

Não havendo uma chave em comum entre as instâncias a integrar, algumas propostas sugerem o uso de atributos comuns. O conteúdo dos atributos comuns podem ser comparados para determinar o grau de similaridade entre duas instâncias. Essa técnica é aplicável a dados semi-estruturados, mas exige considerável intervenção do usuário na especificação do grau de similaridade dos atributos.

Conforme apresentado, uma das técnicas para identificação de instâncias é a submissão de consultas às instâncias de dados, a fim de retornar o valor de uma ou de um conjunto de propriedades que as identifiquem unicamente. Sempre que duas instâncias de um mesmo tipo retornarem o mesmo resultado para a consulta, estas serão consideradas um mesmo objeto do mundo real.

Do mesmo modo, a abordagem proposta neste trabalho visa submeter uma consulta às fontes de dados, a qual retorna o identificador das instâncias. A técnica aqui proposta, combina as técnicas *Consultas definidas pelo usuário* e *Funções booleanas definidas pelo projetista*, conforme apresentado na seção 3.2. O mecanismo proposto consta de:

- utilizar uma função *Skolem* para identificar objetos;
- implementar a função *Skolem* através de consultas em uma linguagem do padrão XML, especificamente *XPath* [BRA2000, W3C99].

Para a resolução de conflitos nos valores das propriedades dos objetos, observa-se que as técnicas estudadas não são de uso generalizado dependendo da aplicação e da propriedade que está sendo considerada. Neste trabalho, procurou-se uma solução genérica e de aplicação para o caso de fontes de dados na WEB. Esta proposta combina duas idéias, *normalização de valores* e *rótulos temporais*.

4 Proposta do Trabalho

Este capítulo apresenta a arquitetura proposta para a materialização de dados XML. São descritos os mecanismos utilizados na geração do esquema lógico relacional, bem como a técnica utilizada para a manutenção incremental do banco de dados. A solução proposta para a identificação das instâncias XML também é apresentada.

4.1 Visão Geral

O objetivo de um sistema de integração de dados é oferecer aos usuários uma interface uniforme de acesso a diferentes fontes de dados. Um sistema de integração de dados também precisa lidar com a heterogeneidade das fontes de dados, as quais, na maioria das vezes, foram projetadas independentemente, utilizando diferentes modelos de dados e diferentes representações para os mesmos conceitos do mundo real [SAL2001].

Uma importante decisão na construção de um sistema de integração de dados na WEB é a escolha da abordagem a ser utilizada: virtual ou materializada. Na abordagem virtual, as informações são extraídas das fontes somente quando consultas são requisitadas. Por outro lado, na abordagem materializada, as informações são recuperadas, integradas e armazenadas em um repositório, de maneira que as consultas podem ser avaliadas diretamente neste repositório, sem a necessidade de acessar as fontes de dados [SAL2001].

Diversas arquiteturas já foram propostas para solucionar o problema de integração de dados. Neste sentido, uma das arquiteturas de destaca: a arquitetura de *data warehouse*. Esta arquitetura está associada à abordagem materializada; geralmente é utilizada quando as fontes de dados sobre as quais o sistema integrado foi construído sofrem alterações não tão freqüentes, quando o custo de acesso às fontes de dados externas é alto e exige onerosos processos de extração. Esta arquitetura ainda mostra-se adequada quando as consultas submetidas pelo usuário requerem porções específicas e previsíveis da informação disponível. Entretanto, esta abordagem requer a manutenção entre os dados armazenados no sistema integrado e os dados das fontes de dados, à medida que estes sofrem alterações.

Os principais tópicos a serem tratados nesta arquitetura são:

- a criação de um modelo conceitual único que descreva o sistema integrado;
- a geração do modelo lógico do sistema integrado;
- a integração das instâncias vindas das diferentes fontes de dados;
- a manutenção do sistema integrado, à medida que as fontes de dados sofrem atualizações.

A seção a seguir apresenta a arquitetura proposta neste trabalho.

4.2 Arquitetura Proposta

A proposta da técnica para materialização de dados XML apresentada neste trabalho é representada por um módulo materializador. Este módulo recebe dados de várias fontes XML, integra e materializa estas informações em um banco de dados relacional. Consultas do usuário final são submetidas diretamente a este banco de dados, utilizando uma linguagem baseada em SQL.

A arquitetura geral do projeto é apresentada na FIGURA 5.

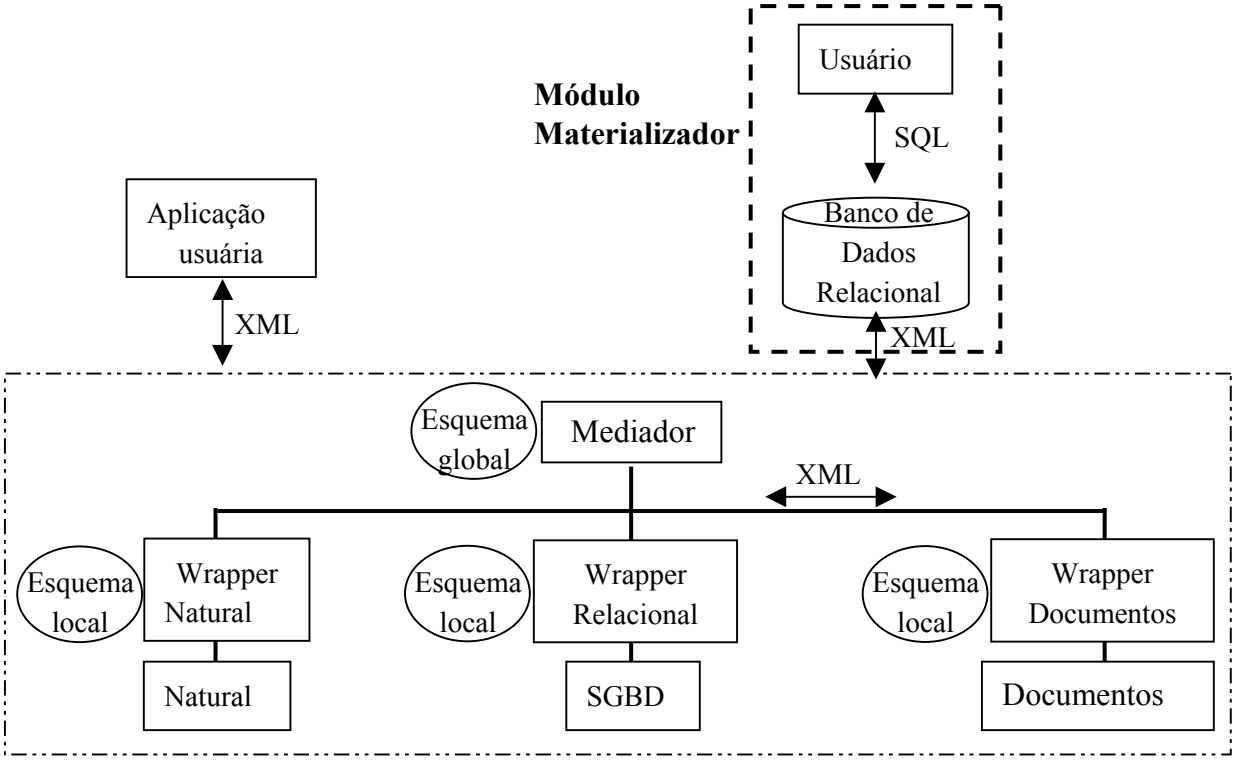

FIGURA 5 - Arquitetura Geral do Projeto

A descrição mais detalhada da arquitetura é apresentada a seguir. O sistema de integração provê acesso a fontes de dados heterogêneas e distribuídas, que podem ser desde sistemas gerenciadores de bancos de dados até simples arquivos de documentos. O primeiro passo consiste em definir o esquema da visão integrada, obtido através da integração dos elementos dos esquemas das fontes de dados [SAL2001]. Além do esquema da visão integrada, algumas informações de descrição das fontes de dados também são requeridas, os metadados. Estes metadados incluem: identificador e localização das fontes de dados, descrições do conteúdo,

informações sobre o esquema, correspondências entre os elementos do esquema integrado e os elementos dos esquemas locais (através de um dicionário de sinônimos). Além disso, também precisa ser definido um módulo tradutor (*wrapper*), que converta os dados das fontes de dados para o modelo de dados comum e as consultas das aplicações em consultas específicas de cada fonte de dados correspondente.

Outro problema que precisa ser solucionado é em relação à heterogeneidade das fontes de dados a serem integradas. Para isso, o sistema de integração usa um modelo de dados comum para representar o conteúdo e a estrutura das fontes de dados. O modelo de dados utilizado é o XML, devido principalmente à sua flexibilidade para representar dados estruturados e semi-estruturados e à facilidade de conversão de dados para este formato.

Este trabalho enfoca o módulo materializador. Este módulo é responsável pelo armazenamento dos dados XML recebidos do *mediador* em um formato XML. Para isso, três tarefas principais precisam ser realizadas: geração do esquema lógico relacional (baseada em uma ontologia), integração dos dados XML extraídos e instanciação/manutenção incremental do banco de dados relacional, conforme mostrado na FIGURA 6. Estes módulos são descritos detalhadamente nas seções 4.3, 4.4 e 4.5.

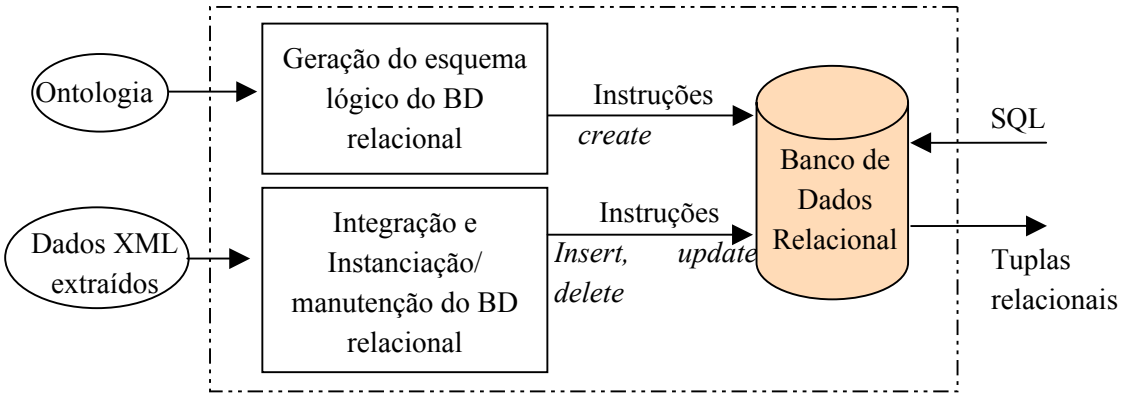

FIGURA 6 - Módulo Materializador

4.3 Proposta de Técnica de Geração do Esquema Lógico Relacional

O módulo de geração do esquema lógico do banco de dados relacional, mostrado na FIGURA 6, é responsável pela especificação de quais tabelas serão criadas, atributos destas tabelas e chaves primárias e estrangeiras. Este módulo utiliza uma ontologia para a geração do esquema lógico relacional. A proposta deste módulo é baseada em [SIL2000].

A estrutura da proposta de [SIL2000] se baseia fundamentalmente na concepção *Closed World Assumption* - CWA, pela qual os documentos considerados estão armazenados em um

banco de dados específico de um determinado sistema. Nesta proposta, é assumido que as instâncias XML possuem identificadores. Esta alternativa se distingue da *Open World Assumption* - OWA, a qual pressupõe uma maior abrangência no conjunto de dados considerado [apud SIL2000, 2000, p.55]. Nesta segunda situação poderiam ser considerados dados da Web, por exemplo.

A ontologia utilizada para este propósito adota um modelo simples, o qual representa conceitos (objetos) do domínio do problema, relacionamentos e cardinalidades entre esses conceitos. Além disso, a ontologia faz a distinção entre *conceitos léxicos* (representáveis diretamente no computador através de cadeias de bits) e *conceitos não léxicos* (sem representação direta em computador). Em outras palavras, conceitos léxicos representam elementos atômicos e atributos de elementos de um documento XML; conceitos não léxicos representam objetos complexos de um documento XML.

A ontologia utilizada é descrita através de um diagrama entidade-relacionamento modificado. O exemplo mostrado na FIGURA 7 representa parte de uma ontologia que descreve a classe de documentos *artigo*.

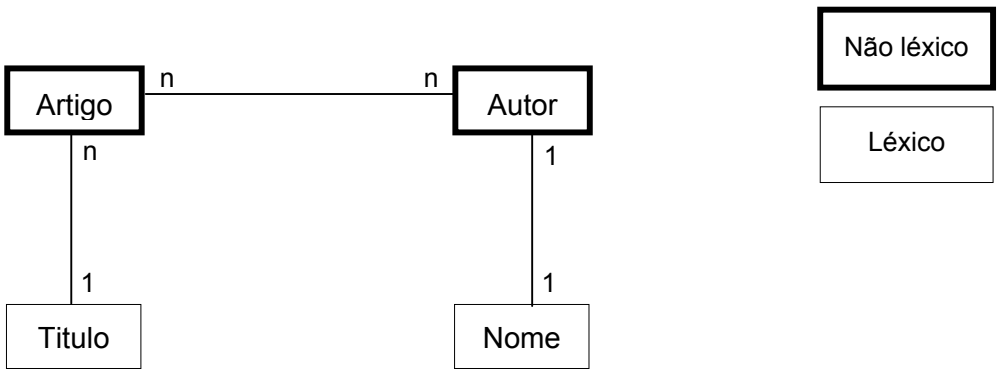

FIGURA 7 - Parte de uma Ontologia

A idéia básica do algoritmo para geração do esquema lógico relacional é baseada em [SIL2000]. Basicamente, elementos não léxicos são mapeados para tabelas, com as colunas destas tabelas sendo definidas através de elementos léxicos. Além disso, de acordo com a cardinalidade dos relacionamentos, podem ser criadas novas tabelas de associação entre elementos, ou mesmo novas colunas para a identificação de chaves estrangeiras [SIL2000a]. Algumas tabelas de metadados são criadas para viabilizar a manutenção incremental do banco de dados relacional.

A seguir, estão detalhados os passos executados para a geração do esquema lógico relacional:

PASSO 1) Inicialmente uma tabela é criada para cada elemento não léxico da ontologia. Desta forma, todos os elementos que não têm representação direta em computador irão gerar tabelas no banco de dados relacional, independente de seus relacionamentos e cardinalidades. Para a ontologia usada como exemplo, seriam criadas as tabelas *autor* e *artigo*

PASSO 2) Criação de uma coluna em cada uma das tabelas geradas para o identificador da instância XML armazenada: *autor (id_autor) ,artigo(id_artigo).* Dessa maneira, garante-se que dentro de um mesmo documento seja possível identificar de forma única cada um dos conceitos existentes. Entretanto, dados XML, como propostos originalmente, não carregam necessariamente a noção de chave primária ou identificador de objeto. Esta característica dificulta o armazenamento dos dados semi-estruturados extraídos, pois as tabelas relacionais se utilizam de campos chaves para associar dados armazenados. Assim, é necessária a adoção de um mecanismo que faça a identificação destas instâncias (descrito na seção 4.4).

PASSO 3) Uma vez definido o campo identificador para cada tabela (identificador do conceito), podem ser criados os demais campos que irão compor a estrutura de cada uma destas tabelas. Estes campos são obtidos a partir dos conceitos léxicos relacionados ao conceito não léxico que originou a tabela. Para cada conceito léxico associado, é criado um campo na tabela: *artigo (id_artigo, titulo), autor (id_autor, nome)*

PASSO 4) A seguir é especificada a associação entre as tabelas criadas, buscando garantir o armazenamento dos relacionamentos entre os conceitos. Na estratégia adotada, optou-se por tratar este problema de acordo com a cardinalidade existente na ontologia entre os conceitos não léxicos.

Para elementos não léxicos relacionados com cardinalidade n:n entre si, é gerada uma nova tabela, a qual armazena os identificadores da cada um dos elementos não léxicos relacionados na ontologia: *artigo_autor (id_artigo, id_autor).*

Para elementos não léxicos relacionados com cardinalidade n:1 entre si, um campo adicional é criado na tabela do primeiro conceito, servindo como chave estrangeira.

Em termos de chaves primárias, o algoritmo prevê duas situações:

- para tabelas geradas a partir da aplicação do PASSO 1: a chave primária é o identificador da instância XML armazenada: *autor (id_autor, nome), artigo (id_artigo, titulo);*
- para tabelas geradas a partir da associação entre elementos não léxicos: a chave primária é formada por todos os campos da tabela criada: *artigo_autor (id_artigo, id_autor).*

Além disso, algumas tabelas de metadados necessitam ser geradas para viabilizar a manutenção incremental do banco de dados. As informações armazenadas nestas tabelas são utilizadas quando do recebimento de uma nova solicitação de consulta. O módulo responsável pela manutenção incremental do banco de dados verifica o rótulo temporal e os conceitos extraídos anteriormente para os documentos já armazenados. Desse modo, o objetivo é evitar, sempre que possível, uma nova operação de extração de dados. Quando a informação desejada já estiver presente e atualizada no banco de dados relacional, a consulta pode ser respondida com o conteúdo do banco de dados relacional, sem o acesso às fontes semi-estruturadas.

Tais tabelas objetivam armazenar:

- a fonte de origem dos dados materializados, bem como a data de última atualização desta fonte (rótulo temporal). Esta data é armazenada na tabela *documentos*, sempre que ocorrer um processo de extração na fonte de dados XML: *documentos (id_fonte, rotulo_temporal).* Na proposta deste trabalho, o campo que identifica a fonte de origem dos dados é a URL onde o documento XML pode ser encontrado. Esta tabela é necessária para que se possa verificar se os dados materializados estão consistentes com os dados das fontes de origem, através da comparação do rótulo temporal desta tabela com o rótulo temporal da fonte de dados;

- os identificadores das instâncias XML já materializadas para um determinado elemento da ontologia, juntamente com sua fonte de origem: *conceitos_materializados (id_fonte, conceito_ontologia, id_instância)*. Esta tabela é necessária para verificar quais fontes de dados referenciam uma determinada instância XML materializada no banco de dados relacional. Esta informação é útil para a remoção de informações no banco de dados. Por exemplo, um determinado registro da tabela *autor* só poderá ser removido se nenhuma fonte de dados XML referencia esta instância;

- uma tabela de sinônimos, a qual armazena todos os sinônimos de nomenclatura das várias fontes para um determinado conceito da ontologia: *sinonimos (id_fonte, conceito_ontologia, nomenclatura_local).*

O algoritmo desenvolvido para a geração do esquema lógico do banco de dados é apresentado abaixo:

```
{
    para cada elemento não léxico faça
    {
        criar tabela para o elemento
        criar coluna de identificação para este elemento
        definir a coluna de identificação do conceito como chave primária

        para cada  par de elementos não léxicos relacionados n:n faça

        {
        se não existe tabela de relacionamento entre esses elementos
        {
        criar tabela de relacionamento entre os elementos
        criar coluna de identificação para o primeiro elemento
        criar coluna de identificação para o segundo elemento
                    definir as colunas de identificação dos conceitos como chave
            primária
            }
        }
        para cada  par de elementos não léxicos relacionados n:1 faça
        {
```

```
            criar coluna de identificação para o elemento relacionado na tabela do
            primeiro elemento
                        definir a coluna de identificação do conceito relacionado como
                  chave estrangeira
            }
            para cada elemento léxico relacionado a um elemento não léxico faça
            {
            criar coluna para o elemento léxico na tabela do elemento não léxico
            relacionado
            }
      }
      criar tabela de documentos
      criar coluna de identificação do documento de origem
      criar coluna para o rótulo temporal do documento extraído
            definir a coluna de identificação do documento de origem como chave
         primária

      criar tabela de conceitos materializados
      criar coluna de identificação do documento de origem
      criar coluna para o elemento extraído
      criar coluna de identificação do elemento extraído
            definir as colunas de identificação do documento de origem, elemento
      extraído e identificação do elemento extraído como chave primária

      criar tabela de identificadores
      criar coluna de identificação do documento de origem
      criar coluna para o elemento extraído
            criar coluna para o armazenamento da consulta XPATH que extrai o
      identificador do elemento
            definir as duas primeiras colunas criadas como chave primária

      criar tabela de sinônimos
      criar coluna de identificação do documento de origem
      criar coluna para o elemento da ontologia
            criar coluna para o armazenamento do sinônimo do elemento da ontologia no
      documento de origem
            definir as colunas de identificação do documento de origem e elemento da
      ontologia como chave primária

}
```

Abaixo, é mostrado o esquema relacional gerado para a ontologia descrita na FIGURA 7:

Artigo (<u>id_artigo</u>, titulo)
Autor (<u>id_autor</u>, nome)
Artigo_autor (<u>id_artigo, id_autor</u>)

Documentos (id_fonte, rotulo_temporal)
Conceitos_materializados (id_fonte, conceito_ontologia, id_instancia)
Sinonimos (id_fonte, conceito_ontologia, nomenclatura_local)
Identificadores (id_fonte, conceito_ontologia, consulta)

4.4 Proposta de Técnica de Identificação de Instâncias XML

Este capítulo apresenta a técnica proposta neste trabalho para identificação de instâncias XML. Também são descritas as linguagens *XPath* e *XSLT*, utilizadas no mecanismo proposto.

4.4.1 Visão Geral

A integração de instâncias vindas de diferentes fontes de dados envolve identificar quais instâncias de dados representam o mesmo objeto do mundo real, bem como resolver ambigüidades de representação deste objeto. A área de BD heterogêneos apresenta este assunto sob os termos *identificação de objetos* e *resolução de conflitos nos valores das propriedades dos objetos*, conforme descrito nas seções 3.2 e 3.3.

No contexto deste trabalho, a identificação de instâncias XML considera que:

1) O*bjetos* são representados por objetos complexos de um documento XML, descritos na ontologia através de conceitos não léxicos (por exemplo: *autor*).

2) *Propriedades de objetos* são representadas por elementos atômicos e atributos de XML, descritos na ontologia através de conceitos léxicos (por exemplo: *nome* na figura da ontologia)
.

Assim, a técnica proposta neste trabalho para identificação de instâncias XML é dividida em *identificação de conceitos não-léxicos* (seção 4.4.4) e *identificação de conceitos léxicos* (seção 4.4.5). O mecanismo aqui proposto consta de usar funções *Skolem* para identificar objetos (seção 4.4.2), e implementar as funções *Skolem* através de consultas em uma linguagem do padrão XML, especificamente *XPath* (seção 4.4.3) e *XSLT* (seção 4.4.5).

O modelo de dados aqui adotado consiste de uma árvore, representando o documento XML, composta por nós e arcos. Nós representam os objetos do documento, enquanto os rótulos são representados nos arcos. Objetos complexos referenciam outros objetos (ex: objeto *autor* na árvore abaixo); objetos atômicos contém valores de um tipo básico, representados por uma cadeia de caracteres (ex: objeto *nome* na árvore abaixo). Por exemplo, o seguinte documento XML é representado na árvore da FIGURA 8.

```
... <artigo>
      <autor>
           <nome>Marcos Santos</nome>
          <email>santos@inf.ufrgs.br</email>
       </autor>
  </artigo>....
```

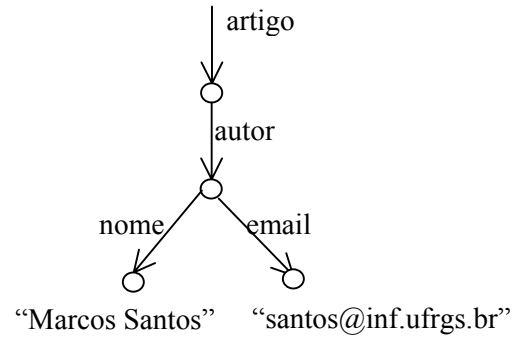

FIGURA 8 - Árvore representativa de um documento XML

4.4.2 Funções *Skolem*

A técnica proposta neste trabalho para identificação de instâncias é baseada no uso de funções *Skolem*. Este termo vem da Lógica de Predicados. No contexto da Lógica de Primeira Ordem, uma função *Skolem* é utilizada para eliminar quantificadores existenciais de predicados sobre um certo domínio. Assim, consegue-se eliminar a dependência entre duas variáveis deste domínio.

Seja o predicado: ∀y (∃x p(x,y))

Este predicado pode ser lido como: para todo *y*, existe um *x* (dependente de *y*), tal que *p(x,y)* é válido. Pelo fato do quantificador existencial (∃) estar dentro do escopo do quantificador universal (∀), é aberta a possibilidade de que *x* depende do valor do *y*.

Por exemplo, a frase ∀x ∃y Vizinho (x, y) para um determinado domínio, diz que cada objeto *b* deste domínio tem pelo menos um vizinho *c*. Assim, pode-se escolher um vizinho para *b*, por exemplo, o mais próximo. Desse modo, para cada *b* existe uma função *f(b)* que devolve o seu vizinho mais próximo. Logo, a frase ∀x ∃y Vizinho (x, y) pode ser simplificada para: ∀x Vizinho(x, f(x))

Este processo de simplificação é chamado *Skolemização*. A função *f* é chamada de função *Skolem* da frase quantificada original. Uma descrição mais formal de função *Skolem* pode ser encontrada em [HEI95].

Em dados orientados a objetos [HUL90], bem como em dados semi-estruturados (XML-QL [DEU99b]), tem-se utilizado o termo função *Skolem* com o propósito de identificação de instâncias. Neste contexto, a função *Skolem* designa uma função que, a partir de um objeto dado, constrói seu identificador. Considerando a seguinte frase:

∀x Vizinho(x, f(x)), onde *f(x)* é uma função *Skolem*. A fim de realizar a identificação de instância, a assertiva acima pode ser interpretada como: para cada instância de dados (x) presente no domínio de interesse, existe uma função (f(x)) que gera o identificador para esta instância. Esta função é dependente única e exclusivamente da instância para a qual se deseja gerar o identificador.

XML-QL faz uso deste tipo de função para transformação de dados, como por exemplo, de uma DTD para outra [DEU99b]. Para ilustrar, considere a seguinte porção de DTD que define um tipo *author:*

```
<!ELEMENT author (firstname, lastname)>
```

Pode-se escrever uma consulta em XML-QL que transforme dados desta DTD em dados que estão de acordo com outra DTD, por exemplo:

```
<!ELEMENT person (lastname, firstname, address?, phone?, publicationtitle*)>
<ATTLIST person ID ID #REQUIRED>
```

A consulta abaixo extrai autores e transforma-os em elementos <person> [DEU99b].

```
WHERE <$><author><firstname>$fn</>
                        <lastname>$ln</>    </>
      </> IN “www.a .b.c/bib.xml”,
CONSTRUCT <person ID=PersonID ($fn, $ln)
                          <firstname>$fn</>
                          <lastname>$ln</> </>
```

Sempre que um elemento <person> é produzido, seu ID associado é *PersonID ($fn, $ln)*. *PersonID* é uma função *Skolem*, e seu propósito é gerar um novo identificador para cada valor distinto de *firstname* e *lastname*. Entretanto, XML/QL não especifica como as funções *Skolem* são implementadas. Na proposta deste trabalho, utiliza-se a linguagem *XPath* na implementação destas funções.

4.4.3 A Linguagem *XPath*

O padrão *XPath*, proposto pelo W3C, é uma linguagem que permite referenciar partes de um documento XML. Foi projetada para ser utilizada pelos padrões XSLT e *XPointer*

[W3C99]. A linguagem fornece facilidades básicas para manipulação de *cadeias de caracteres*, números e booleanos [W3C99]. Em tratando-se de manipulação de cadeias de caracteres, *XPath* propõe um variado conjunto de funções que possibilitam: extração de subcadeias de caracteres contidas em elementos, concatenação, conversão para maiúsculo/minúsculo, eliminação de espaços em branco, etc.

O padrão modela um documento XML como uma árvore de nós. O construtor sintático básico em *XPath* é a expressão: uma cadeia de caracteres que consiste de instruções para selecionar um elemento, atributo, outras estruturas de marcação, ou uma cadeia de texto [BRA2000]. Elas identificam um item por sua localização na estrutura hierárquica do documento. Por exemplo, a expressão *autor* seleciona filhos do elemento corrente que tem o nome "autor". Por elemento corrente, entende-se a posição atual na árvore do documento XML. Um outro exemplo é a expressão *livro/titulo*, a qual seleciona o título de um elemento "livro".

Uma expressão *XPath* é uma série de passos de localização, separadas por /. Cada passo seleciona um conjunto de nós em relação ao nó corrente. Estes nós tornam-se o nó (s) corrente para o passo seguinte. O conjunto de nós selecionados pela expressão são os nós restantes após ter processado cada passo. Por exemplo, a expressão child::Part consiste de um único passo. Este passo seleciona o conjunto de todos os elementos Part que são filhos do nó corrente. A expressão parent::node()/child::Part tem dois passos. O primeiro passo (parent::node()) seleciona um único nó, o pai do nó corrente. O segundo passo (child::Part) seleciona todos os elementos Part que são filhos desse nó. Assim, a expressão identifica todas os filhos Part do pai do nó corrente. Ou seja, seleciona todos os irmãos Part do nó corrente, incluindo o nó corrente.

Um passo em uma expressão *XPath* consiste em três partes: um eixo, um teste de nó, e zero ou mais predicados. O eixo especifica a direção para mover-se na árvore original. Por exemplo, o eixo child seleciona todos os nós filhos. O eixo parent seleciona o nó pai. O eixo self seleciona o nó corrente. O eixo descendant seleciona todos os nós descendentes. E assim por diante.

Um teste de nó testa se os nós encontrados ao longo do eixo especificado devem ser selecionados para o passo seguinte. Por exemplo, em child::Part, child é o eixo e Part é o teste de nó. Um nó filho satisfaz ao teste de nó se ele for um elemento com o nome Part. Há testes de nó que verificam o nome de elemento e atributo, testam se o nó é um texto, comentário, ou uma instrução de processamento.

Um predicado é uma expressão que filtra os nós selecionados pelo teste de nó. Um predicado tem a forma de uma expressão de igualdade (=,! =, >, <, > =, < =). Por exemplo [BOU2001b]:

[self::text()="123"] testa se o texto do nó corrente é "123"; [child::PartNumber/self::text()="123"] testa se o nó corrente tem um filho PartNumber com texto "123". Desse modo, a expressão child::PartNumber[self::text()="123"] seleciona todas os filhos PartNumber do nó corrente com texto "123"; child::Part[child::PartNumber/self::text()="123 "] seleciona todas os elementos Part que são filhos do nó corrente e tem um elemento PartNumber com "123".

O resultado que se obtém ao avaliar uma expressão é um conjunto de nós selecionados. Expressões podem conter, recursivamente, outras expressões usadas para filtrar conjuntos de nós, através do uso de predicados: autor[email = “santos@inf.ufrgs.br”]

A avaliação da expressão acima retorna o autor cujo endereço eletrônico seja o especificado na cadeia de caracteres.

Além disso, o padrão *XPath* propõe um variado conjunto de funções para o tratamento de testes de posição de elementos (selecione o primeiro autor), valores de atributos (selecione o atributo *edição* do elemento livro), manipulação de *cadeias de caracteres* (selecione o parágrafo que inicia com “Devido a isso”), entre outras. Uma descrição completa de *XPath* pode ser encontrada em [W3C99]. Basicamente, a proposta deste trabalho faz uso de expressões e funções que possibilitam a manipulação de cadeias de caracteres: extração de sub-cadeias de caracteres contidas em elementos, concatenação, conversão para maiúsculo/minúsculo, eliminação de espaços em branco e assim por diante.

4.4.4 Técnica Proposta para Identificação de Elementos Não Léxicos

O mecanismo aqui proposto consta de:

- usar funções *Skolem* para identificar objetos;
- implementar as funções *Skolem* através de consultas em uma linguagem do padrão XML, especificamente *XPath* [BRA2000, W3C99].

A seguir são apresentados vários exemplos de funções *Skolem* codificadas em *XPath*.

Suponha que o primeiro e último nomes de um autor sejam suficientes para identificar unicamente uma instância de dados deste tipo. Assim, a consulta sobre este elemento deve retornar estas duas cadeias de caracteres para fins de identificação. Sempre que os mesmos valores forem retornados pela consulta, considerar-se-á que se trata de um mesmo objeto do mundo real; retornos diferentes simbolizam duas instâncias de dados que modelam dois objetos distintos.

A função *Skolem* ainda pode, por exemplo, fazer a concatenação das duas *cadeias de caracteres*, convertê-las para maiúsculo, fazer a eliminação de espaços em branco (se presentes), mapeá-las para um valor numérico, e assim por diante; o resultado final seria o identificador desta instância (ID).

Considerando o mesmo caso, suponha duas instâncias de dados XML, vindas de fontes diferentes, com estruturas distintas:

```
... <author>
     <firstname>Marcos</firstname>
     <middlename>Alberto</middlename>
     <lastname>Santos</lastname>
   </author>...
```

```
... <person>
      <lastname>Santos</lastname>
      <firstname>Marcos</firstname>
    </person>...
```

Considerando que a informação contida nos rótulos <*firstname*> e <*lastname*> é suficiente para identificar a instância, a idéia é submeter uma consulta à cada fonte de dados que retorne o conteúdo destes rótulos, concatene as duas cadeias de caracteres e converta-as para maiúsculo, gerando o seguinte identificador: ID = "MARCOSSANTOS". De posse do identificador gerado, agora tem-se o conhecimento de que as duas instâncias XML referem-se ao mesmo objeto do mundo real. Logo, esta instância será considerada uma única vez no momento da integração dos dados.

O exemplo acima foi propositadamente simplificado para fins de compreensão. A informação necessária para identificação estava diretamente representada nos rótulos <*firstname*> e <*lastname*>. Uma consulta simples em alguma linguagem de consulta para dados XML retornaria os valores desejados. Entretanto, em se tratando de dados semi-estruturados, isso não é sempre o que ocorre. A informação necessária para identificação de instâncias pode estar encravada na cadeia de caracteres de um determinado *rótulo*, sendo necessário então um pré-processamento a fim de extraí-la antes da geração do identificador. Ainda em outro caso, pode ser necessário extrair subpartes de vários rótulos do documento XML, concatená-las e convertê-las para um formato específico. Por exemplo, considere que um evento científico seja identificado por seu nome e o ano de realização. Tal informação encontra-se disponível na seguinte instância XML:

```
... <artigo>
      <autor>
   <nome>Marcos Alberto Santos</nome>
   <email>santos@inf.ufrgs.br</email>
      </author>
      <titulo>Caching XML Data</titulo>
      <evento>Simposio Brasileiro          de Banco de Dados, RJ, julho  2001</evento>
   </artigo>
```

Para gerar o identificador deste evento, é necessário extrair o nome do evento e o ano de sua realização. Embora esta informação seja visível ao ser humano, ela não está representada de maneira diretamente manipulável por uma linguagem de consulta à dados XML. O nome e o ano de realização do evento estão embutidos no rótulo <evento>, e devem ser extraídos a fim de gerar o identificador da instância. Para atingir este objetivo, utiliza-se o padrão *XPath*, apresentado na seção 4.4.3.

Retornando ao caso da identificação de um evento pelo seu nome e ano de realização. O nome do evento encontra-se disponível no início do *rótulo* e estende-se até a localização da primeira vírgula. Suponha que este seja um padrão constante em todos os rótulos <evento> de um determinado documento XML. Pode-se tirar vantagem deste fato, e através do uso de *XPath*, extrair a informação do nome do evento, utilizando a seguinte função:

string substring_before(string, string).

Esta função retorna a *sub-cadeia* do primeiro argumento que precede a primeira ocorrência do segundo argumento no primeiro argumento. Por exemplo:

substring_before("Simposio Brasileiro de Banco de Dados, RJ, julho 2001", ",")

retorna "Simposio Brasileiro de Banco de Dados" (os vários espaços em branco são propositais). O ano do evento também é necessário para identificação, então suponha que este encontre-se sempre disponível nas quatro últimas posições do *rótulo* <evento>. Para extraí-lo, utiliza-se a função

substring("Simposio Brasileiro de Banco de Dados, RJ, julho 2001",
string_length("Simposio Brasileiro de Banco de Dados, RJ, julho 2001")-3,4)

Esta função retorna a sub-cadeia do primeiro argumento, começando na posição especificada no segundo argumento com o tamanho especificado no terceiro argumento. Sabendo-se que a função *string_length* retorna o número de caracteres na cadeia passada como parâmetro, a expressão acima retorna "2001" (para otimizações no uso destas funções, ver [W3C99]). Dispondo agora do nome do evento e do seu ano de realização, pode-se gerar o identificador da instância. A função *Skolem* pode ainda realizar um processamento sobre o nome e o ano do evento. O tipo deste processamento é definido pelo administrador responsável pela integração dos dados. Vamos considerar que o identificador a ser gerado é uma cadeia de caracteres, resultante da concatenação dos dois argumentos, previamente convertidos para maiúsculo e sem a presença de múltiplos espaços em branco entre as palavras. Também fazendo uso das funções do padrão *XPath*, são mostradas abaixo os passos a serem realizados pela função *Skolem.* Os resultados parciais são atribuídos à variáveis temporárias para melhor compreensão:

1) eliminação de espaços em branco:

a = normalize-space ("Simposio Brasileiro de Banco de Dados")

a = "Simposio Brasileiro de Banco de Dados"

Esta função remove espaços em branco no início e no final da cadeia de caracteres passada como argumento, assim como também substitui vários espaços em branco para uma única ocorrência. Por espaços em branco entende-se caracteres de espaço propriamente ditos, retornos de carro (*carriage return*), quebras de linha (*line feeds*) e tabulações (*tabs*).

2) conversão para maiúsculo:

b=translate(a, “abcdefghijklmnopqrstuvwxyz”, “ABCDEFGHIJKLMNOPQRSTUVWXYZ”)

b = “SIMPOSIO BRASILEIRO DE BANCO DE DADOS”

O identificador final gerado será o resultado da concatenação da cadeia de caracteres *b* com o ano do evento:

ID = concat (b, “ ”, “2001”)
ID = “SIMPOSIO BRASILEIRO DE BANCO DE DADOS 2001”

Realizando todos os passos paralelamente, a função responsável pela geração do identificador de um evento em uma determinada fonte XML que segue os padrões acima considerados na extração teria o seguinte estilo:

ID = (concat(translate(normalize-space(substring_before(evento, “,”))),
“abcdefghijklmnopqrstuvwxyz”, “ABCDEFGHIJKLMNOPQRSTUVWXYZ”), “ ”,
substring(evento, string_length(evento)-3,4))

ID = “SIMPOSIO BRASILEIRO DE BANCO DE DADOS 2001”

Deve-se salientar que a linguagem *XPath* inclui uma ampla variedade de funções que permitem a manipulação de documentos XML. Essa ampla flexibilidade permite os mais variados tratamentos de instâncias XML, e o identificador a ser gerado pode ser resultado de processamentos complexos na estrutura do documento.

No exemplo trabalhado, supôs-se que um evento era suficientemente identificado por seu nome e ano de realização. Tal suposição é advinda de um estudo do domínio do problema, e fica a cargo do projetista do sistema que faz a integração das fontes. Esse conhecimento externo do domínio é fundamental para que se faça a identificação das instâncias.

Conforme já dito, dados XML são semi-estruturados. A informação necessária para identificação de um determinado objeto (por exemplo, evento) pode estar disposta de maneira variada entre diversas fontes. Dessa maneira, deve ser escrita uma função *Skolem* diferente a cada fonte quando a estrutura não for a mesma. Por exemplo, supondo que em um outro documento XM, o nome do evento e o seu ano de realização estivessem explicitamente armazenados em rótulos separados, conforme mostrado abaixo:

```
... <evento>
  <nomeEvento>Simposio Brasileiro de Banco de Dados</nomeEvento>
  <anoEvento>2001</anoEvento>
  </evento>
```

A função *Skolem* para geração do identificador de evento para esta fonte seria do estilo:

ID = (concat(translate(nomeEvento, “abcdefghijklmnopqrstuvwxyz”, “ABCDEFGHIJKLMNOP
QRSTUVWXYZ”)), “ ”, anoEvento)

ID = “SIMPOSIO BRASILEIRO DE BANCO DE DADOS 2001”

Note que, apesar da estrutura de documentos XML variar entre as fontes, é assumido que dentro de uma fonte ela mantém-se constante. No exemplo acima, é considerado que a informação necessária para a geração do identificador estava disponível nos rótulos <nomeEvento> e <anoEvento>. Se esta regra não for satisfeita em todo o documento XML, o identificador gerado não será o esperado.

Das considerações acima, pode-se constatar que o administrador do sistema integrado é o responsável pela definição de qual informação é necessária para identificação das entidades a serem integradas. Além disso, uma função *Skolem* deve ser escrita à cada fonte XML (supondo que suas estruturas, rótulos e disposições das informações necessárias não sejam as mesmas) para cada tipo de objeto a ser identificado; todas essas funções devem retornar o mesmo identificador, quando tratarem da mesma entidade do mundo real.

O modelo de dados inicialmente utilizado (mostrado na FIGURA 8) é estendido, a fim de suportar o identificador criado para as instâncias XML. Desse modo, é criado um nó ID para cada objeto complexo do documento XML. Este nó contém o identificador criado para a instância. Considerando o seguinte documento XML:

```
... <artigo>
        <autor>
   <nome>Marcos Santos</nome>
        </author>
    </artigo>...
```

Supondo que o identificador gerado para objetos do tipo *autor* seja definido pelo primeiro nome, ter-se-ia a seguinte função *XPath* responsável pela extração do identificador:

ID = (translate (substring_before(nome, “ ”)), “abcdefghijklmnopqrstuvwxyz”, “ABCDEFGHIJKLMNOPQRSTUVWXYZ”)

ID = “MARCOS”

Desse modo, a árvore retornada do processo de extração para o documento XML acima seria representado na árvore da FIGURA 9:

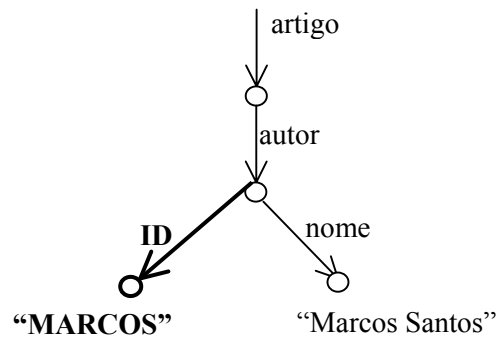

FIGURA 9 - Árvore representativa de um documento XML

4.4.5 Técnica Proposta para Identificação de Elementos Léxicos

Após duas instâncias serem identificadas como representando um mesmo objeto do mundo real (através da identificação de conceitos não léxicos), os valores de suas propriedades podem apresentar valores conflitantes, como por exemplo, dois endereços diferentes para um mesmo autor. Neste trabalho considera-se que propriedades de objetos são representadas em XML por elementos atômicos e atributos, ou seja, pelas folhas da árvore representativa de uma instância XML (descritos na ontologia por conceitos léxicos).

Este tipo de conflito é tratado neste trabalho como identificação de conceitos léxicos. Por exemplo, no caso de dois endereços para um mesmo autor, como identificar que estes endereços são diferentes? Para o tratamento deste tipo de conflito, várias abordagens foram propostas na literatura para bases de dados estruturadas. Estas técnicas foram apresentadas na seção 3.3.

Neste trabalho, procurou-se uma solução genérica e de aplicação para o caso de fontes de dados na WEB. Esta proposta combina duas idéias, *normalização de valores* e *rótulos temporais*.

Normalização de valores

Por *normalização de valores* entende-se a transformação dos valores originais das fontes em um valor "normalizado" que pode ser comparado com o valor "normalizado" proveniente de uma outra fonte. Exemplificando, caso, em uma fonte, dias da semana apareçam na forma de seus nomes em Português e em outra fonte codificados na forma de números, através da normalização de valores, os dados de uma das fontes, ou de ambas, são convertidos para um conjunto canônico de valores. A proposta é de que a função de normalização seja codificada na linguagem XSLT [W3C99].

XSLT

De acordo com a especificação do W3C [W3C99], XSLT é uma linguagem projetada primeiramente para transformar um documento XML em outro documento XML. Entretanto, XSLT é capaz de transformar XML para HTML e muitos outros formatos baseados em texto. [KAY2000] define XSLT como uma linguagem para transformar a estrutura de um documento XML.

Transformações XSLT podem ser utilizadas na conversão e publicação de dados. Pode ser utilizada para extrair dados, reordená-los, transformar atributos em elementos e vice-versa, e várias outras tarefas similares. Além disso, procedimentos mais complexos, escritos em

outras linguagens (como Java, por exemplo) podem ser invocados de dentro do *stylesheet* para realizar transformações nos dados.

XSLT utiliza *XPath* como uma sublinguagem. Geralmente, *XPath* é utilizada para identificar partes do documento de entrada a serem processadas. As transformações a serem aplicadas no documento de entrada são especificados em um *stylesheet*. Um *stylesheet* é formado por um conjunto de regras *template.* Transformações são executadas de acordo com tais regras e cada regra equivale a um tipo de elemento no documento de entrada utilizando-se expressões *XPath.*

Como exemplo, considerando o seguinte documento XML de entrada:

```
<listalivros>
        <livro>
                <titulo>Data on the Web</titulo>
                <autor>Abiteboul, Serge</autor>
                <ano>2000</ano>
        </livro>
</listalivros>
```

Aplicando-se o *stylesheet* a seguir:

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

<xsl:template match="livro">
<Book>
<xsl:apply-templates/>
</Book>
</xsl:template> ... ...

<xsl:template match="autor">
<Author>
<xsl:value-of select="."/>
</Author>
</xsl:template>

<xsl:template match="titulo">
<Title>
<xsl:value-of select="."/>
</Title>
</xsl:template>
```

Tem-se o seguinte documento XML gerado:

```
<BookList>
        <Book>
                <Author>Abiteboul, Serge</Author>
                <Title>Data on the Web</Title>
        </Book>
</BookList>
```

XSLT é utilizado na proposta deste trabalho para normalização de valores. Conforme já mencionado anteriormente, o objetivo é a transformação dos valores originais das fontes em um valor "normalizado" que possa ser comparado com o valor "normalizado" proveniente de uma outra fonte. A comparação destes valores normalizados determina se as duas instâncias referem-se a uma mesma entidade do mundo real.

Considerando as seguintes fontes XML:

Fonte1.xml

```
... <author>...
      <address>
         <state>Rio Grande do Sul</state>
      </address>
   </author>...
```

Fonte2.xml

```
... <autor>
        <endereco>RS</address>
    </autor>...
```

Semanticamente, as cadeias de caracteres *Rio Grande do Sul* e *RS* expressam o mesmo significado. Ambas referem-se à mesma UF. Entretanto, estão representadas diferentemente nas duas fontes de dados. Considerando que trata-se de um mesmo autor, a extração de dados destas fontes retornará dois valores conflitantes para a entidade UF. Utilizando-se XSLT, pode-se gerar um valor normalizado para esta entidade.

A informação de que *Rio Grande do Sul* e *RS* são semanticamente iguais é adquirida através do administrador do banco de dados. Deste modo, pode ser escrito um *stylesheet* que produza o valor normalizado. Sempre que for encontrada a primeira das cadeias de caracteres (*Rio Grande do Sul*), será aplicada uma transformação XSLT, devolvendo como resultado um outro documento XML com o valor normalizado.

O *stylesheet* que faz esta transformação nos dados vindos da fonte *fonte1.xml* é mostrado a seguir:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output indent="yes"/>

  <xsl:template match="/">
```

```
   <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="text()">

  </xsl:template>

 <xsl:template match="author">
   <Autor>
     <xsl:apply-templates/>
   </Autor>
  </xsl:template>
  <xsl:template match="address">
   <Endereco>
     <xsl:apply-templates/>
   </Endereco>
  </xsl:template>

<xsl:template match="state">
   <Estado>
     <xsl:choose>
 <xsl:when test=".='Rio Grande do Sul'">RS</xsl:when>
 <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
</xsl:choose>
    </Estado>
  </xsl:template>
</xsl:stylesheet>
```

O processamento deste *stylesheet* produz o seguinte documento XML:

```
... <autor>...
      <endereco>
          <estado>RS</estado>
      </endereco>
   </autor>...
```

Com isso, tem-se um valor normalizado para a entidade UF. Apenas o valor normalizado é considerado no momento da materialização das informações no banco de dados relacional. Na ocorrência de conflitos nos valores normalizados, utiliza-se a informação da fonte que tiver o *rótulo temporal* mais atual (apresentado na seção a seguir).

Rótulos temporais

Parte-se do pressuposto que cada fonte de dados possui um *rótulo temporal* que identifica sua última atualização. O mecanismo proposto consta, simplesmente de, em caso de conflito do valor normalizado, utilizar o valor que possui o rótulo temporal mais recente.

Supondo que o autor identificado por "MARCOSSANTOS" esteja representado em duas fontes de dados, como mostrado abaixo:

Fonte1.xml

```
... <nome>Marcos A . Santos</nome>
    <endereco>Rua x, 110</endereco> ...
```

Fonte2.xml

```
... <firstname>Marcos</firstname>
... <lastname>Santos</lastname>
    <address>Rua y, 200</address> ...
```

Considerando ainda que a fonte1.xml tem como data de última modificação (*rótulo temporal*) a data "01/01/2000" e a fonte 2.xml a data "03/07/2000". Sendo assim, o módulo responsável pela integração das fontes verifica os *rótulos temporais* das fontes que referenciam esse autor e considera o endereço "Rua y, 200" como o correto, já que a fonte de dados 2 é a que foi modificada por último. As informações de *rótulos temporais* das fontes e de quais fontes referenciam quais instâncias integradas são necessárias para a adoção desta técnica.

4.4.6 Resumo

A identificação de instâncias XML dessa proposta pode ser esquematizada como segue:

1) Objetos, representados por objetos complexos de um documento XML são identificados por funções *Skolem*, codificadas na forma de consultas *XPath* às fontes de dados originais.

2) Propriedades de objetos, representadas por elementos atômicos e atributos de XML são, por *default*, identificadas pelo seu conteúdo. No entanto, um valor normalizado pode ser utilizado como identificador, utilizando a linguagem XSLT. Na ocorrência de conflitos nos valores normalizados, utiliza-se a informação da fonte que tiver o *rótulo temporal* mais atual.

Na forma em que se encontra a proposta é exigido do usuário o conhecimento e a utilização de XSLT/*XPath*. Para tornar a proposta utilizável por usuários que não dominam estas técnicas é necessária a construção de uma interface gráfica na qual o usuário possa escolher entre esqueletos pré-construídos de consultas *XPath*. Estes esqueletos podem ser definidos a partir de uma classificação dos tipos de identificadores que são usados mais freqüentemente em fontes XML. Exemplos poderiam ser o conteúdo de uma propriedade, a concatenação do conteúdo de várias propriedades, e assim por diante.

4.5 Instanciação e Manutenção Incremental da Visão Materializada

Este capítulo apresenta as políticas de manutenção de visões materializadas encontradas na literatura e apresenta o mecanismo proposto neste trabalho.

4.5.1 Visão Geral

À medida que as fontes de dados sofrem modificações, a visão materializada definida sobre estas fontes deve ser atualizada. O processo para atualizar uma visão materializada em resposta às mudanças nos dados fonte é chamado *manutenção da visão*. Basicamente, existem duas opções para a manutenção da visão materializada [SAL2001]:

- rematerialização da visão: o conteúdo da visão materializada é descartado e a visão é novamente materializada de acordo com os novos dados das fontes de dados;
- manutenção incremental: mudanças nos dados das fontes de dados locais são propagadas incrementalmente para a visão materializada.

Em muitos casos, é desvantajoso reprocessar todo o conteúdo da visão, já que geralmente apenas uma parte dos dados é modificada. Desse modo, processa-se na visão apenas as alterações realizadas nas fontes (manutenção incremental).

O problema de manutenção incremental de visões tem sido bastante estudado no modelo relacional [GUP99] e, recentemente, no modelo orientado a objetos [ALI2000]. Manutenção incremental de visões materializadas para dados semi-estruturados tem sido pouco estudada [ABI98, LIE99]. Além disso, os mecanismos propostos para manutenção incremental de visões para dados semi-estruturados geralmente baseiam-se no fato de que a visão está materializada em sistemas proprietários [LIE99, McH97]. Desse modo, o processo de manutenção pode ser facilitado.

Neste trabalho, a visão definida sobre dados XML é materializada em um sistema de banco de dados relacional; além disso, não é exigido que as fontes de dados possuam identificadores em suas instâncias. Muitos dos algoritmos encontrados na literatura para manutenção de visões baseiam-se na existência de chave primária ou recursos de gatilhos nas fontes, os quais notificam o sistema integrado, quando ocorrem atualizações [GUP99]. As fontes de dados XML não possuem recursos de gatilhos ou algum evento de monitoramento de atualização sobre os dados fonte. Desse modo, muitas das técnicas existentes são inviáveis de serem utilizadas no contexto deste trabalho.

Para a manutenção incremental da visão materializada, o presente trabalho necessita:

- a data de última modificação das fontes de dados sobre as quais a visão é definida

- ter acesso a todo o conteúdo das fontes de dados base, de modo que as instâncias possam ser extraídas e atualizadas na visão materializada, quando necessário.

4.5.2 Política de Manutenção de Visões Materializadas

O processo de manutenção incremental deve decidir quando a visão será atualizada. Visões podem ser atualizadas dentro da mesma transação responsável pela atualização dos dados base; em uma outra situação, a manutenção pode ser realizada mais tarde. O primeiro caso é referenciado como manutenção imediata e o segundo caso como manutenção adiada de visões [GUP92]:

Visões Imediatas

A visão é atualizada imediatamente após uma atualização nos dados base sobre os quais a visão é definida, como parte da transação que atualiza as fontes de dados. Esta técnica permite que as consultas possam ser respondidas rapidamente, pois a visão materializada vai estar sempre consistente com os dados fonte. Em contrapartida, aumenta o tempo das transações de atualização, já que cada transação de atualização implica em propagar as mudanças dos dados fonte para a visão materializada.

Visões Adiadas

Nesta técnica, a visão é atualizada em uma transação separada, separada da transação que atualiza as fontes de dados. Diferentes políticas podem ser definidas:

- "preguiçosa adiada": a visão é atualizada tão tarde quanto seja possível, desde que esteja garantindo que todos os resultados das consultas submetidas estejam consistentes com os dados base. Em outras palavras, a visão não precisa estar totalmente consistente com os dados fonte sobre os quais ela foi definida, mas as consultas sobre a visão têm de ser respondidas como se ela estivesse consistente. A desvantagem desta técnica é que ela impõe um custo significativo nas transações de consulta, já que a consulta pode ter que esperar até que a visão materializada seja atualizada;
- periódica adiada: a visão é modificada periodicamente em intervalos pré-estabelecidos, em uma transação especial de atualização. Esta técnica permite consultas rápidas e não aumenta o tempo das atualizações; a desvantagem é que as consultas submetidas à visão podem trazer resultados inconsistentes em relação aos dados fonte;

- atraso forçado: a visão é atualizada depois de um certo número de modificações nos dados. Assim como a abordagem *periódica adiada,* a desvantagem é que as consultas submetidas à visão podem trazer resultados inconsistentes com os dados fonte.

Na arquitetura do projeto onde este trabalho está inserido, as instâncias de dados XML são materializadas no banco de dados relacional sob demanda das consultas do usuário. Desse modo, consultas subseqüentes que solicitem conceitos da ontologia já materializados anteriormente e cujo conteúdo esteja atualizado em relação às fontes de dados, podem ser respondidas sem uma nova extração de dados XML nas fontes. Portanto, estar-se-á utilizando a política "preguiçosa adiada".

Mecanismo proposto para a Manutenção Incremental

O mecanismo de manutenção incremental do banco de dados relacional é descrito na FIGURA 10.

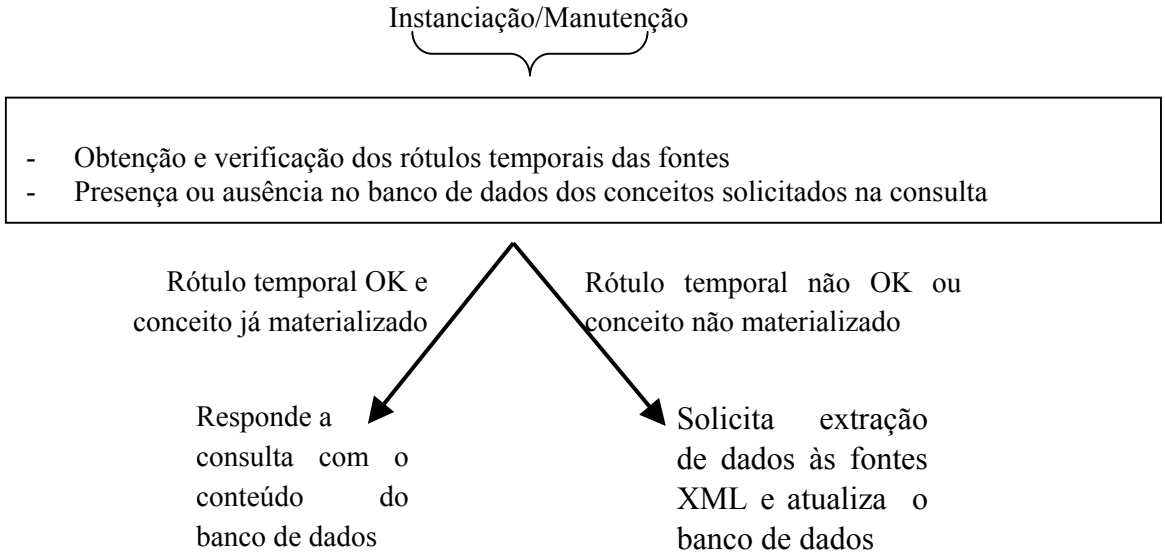

FIGURA 10 - Mecanismo de Atualização Incremental do banco de dados

Ao receber uma consulta do usuário, solicita-se à cada fonte de dados XML sua data de última modificação. Essa data, retornada da fonte, é então comparada com o campo *rotulo_temporal* da tabela *documentos*, o qual armazena a última data em que esta fonte tinha sido modificada, quando alguma instância foi extraída e materializada anteriormente. Se a comparação destas duas datas retornar uma igualdade, o próximo passo é verificar se o conceito solicitado na consulta já está materializado no banco de dados; conforme já mencionado, os dados são materializados sob demanda das consultas feitas pelo usuário. Essa verificação é realizada, consultando-se a tabela *conceitos_materializados*, a qual armazena

quais conceitos já estão materializados para uma determinada fonte de dados XML. Se estes conceitos já estiverem materializados, então a consulta pode ser respondida com o conteúdo do banco de dados.

Caso os dados materializados estejam desatualizados em relação às fontes de dados, ou os conceitos da ontologia solicitados na consulta ainda não estejam materializados, uma extração de dados é realizada e, posteriormente, a inserção ou atualização destes dados no banco de dados relacional.

O documento XML, retornado do processo de extração, é representado como uma árvore. Essa árvore é composta por nós e arcos. Nós representam os objetos do documento, enquanto que os rótulos são representados nos arcos. Objetos complexos (conceitos não léxicos) referenciam outros objetos; objetos atômicos (conceitos léxicos) contém valores de um tipo básico, representados por uma cadeia de caracteres, conforme mostrado na FIGURA 11.

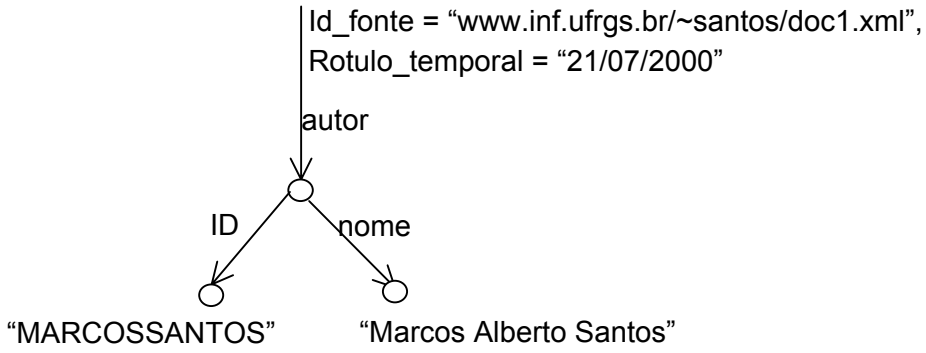

FIGURA 11 - Árvore representativa do documento XML, retornada do processo de extração

A nomenclatura dos nós da árvore se dá de acordo com os termos da ontologia. O problema de sinônimos entre os termos da ontologia e os das fontes de dados podem ser solucionados através de um dicionário de termos; nesta proposta, a tabela *sinonimos* é utilizada para este propósito. O identificador gerado pela função *Skolem,* utilizando *XPath,* é representado no nó *ID*; a data de última alteração da fonte de dados também é retornada do processo de extração.

De posse da árvore recebida do processo de extração, a manutenção do banco de dados segue o seguinte mecanismo. Através da comparação dos nós *ID* com os identificadores das instâncias já materializadas no banco de dados para esta fonte, operações de *insert, update* ou *delete* podem ser realizadas:

CASO 1) Identificador presente na árvore e ausente no banco de dados: operação *insert* deste objeto no banco de dados. No caso da árvore da FIGURA 11, um registro é inserido na tabela *autor.*

CASO 2) Identificador presente no banco de dados e ausente na árvore recebida da extração: operação *delete* no banco de dados. Registros serão removidos das tabelas

resultantes da associação entre conceitos não léxicos (como por exemplo, *artigo_autor*) e da tabela *conceitos_materializados* . Registros das tabelas originadas a partir de conceitos não léxicos (*autor, artigo*) só serão removidos se nenhuma outra fonte de dados referenciar os identificadores das instâncias armazenadas nestes registros. Por exemplo, o autor identificado por "MARCOSSANTOS" só é removido da tabela autor se nenhuma fonte referencia esse identificador de instância. O controle de quais fontes referenciam quais instâncias de dados materializadas pode ser feito através da tabela *conceitos_materializados.*

CASO 3) Identificador presente na árvore e presente no banco de dados: possível operação de *update* no banco de dados. Na ocorrência de conflitos nos valores das propriedades para o mesmo objeto do mundo real (exemplo: dois endereços diferentes para um mesmo autor), utiliza-se a informação da fonte que tiver o *rótulo temporal* mais atual. Supondo que a informação já materializada tenha vindo da fonte de dados com rótulo temporal 10/10/2000, e a informação recém extraída tenha vindo de outra fonte de dados com rótulo temporal 12/11/2000. O valor do atributo a ser mantido no banco de dados será o valor vindo da fonte atualizada mais recentemente. Ou seja, o valor da propriedade a ser mantido no banco de dados, neste caso, é o da segunda fonte de dados. Operações de *update* são realizadas nas tabelas criadas a partir de conceitos não léxicos (*autor, artigo*).

Além disso, as tabelas *documentos* e *conceitos_materializados* devem ser mantidas, sempre que ocorrer uma nova extração.

O mecanismo aqui apresentado para manutenção da visão materializada é disparado sempre que os dados já materializados estejam desatualizados em relação às fontes de dados (casos 2 e 3), ou os conceitos da ontologia solicitados na consulta ainda não estejam materializados (caso 1). Em ambos, uma nova extração de dados é feita. Posteriormente, é realizada a inserção ou atualização destes dados no banco de dados relacional.

O algoritmo desenvolvido para a manutenção incremental da visão materializada é descrito a seguir.

⇒ o usuário submete uma consulta no banco de dados relacional, solicitando determinados conceitos da ontologia
⇒ o sistema solicita às fontes de dados XML a última data de atualização
⇒ a última data de atualização das fontes de dados é comparada com o atributo rotulo_temporal da tabela Documentos para estas fontes
⇒ se o rótulo temporal não for encontrado na tabela *Documentos* para a fonte de dados (rotulo temporal ausente significa que a fonte ainda não foi materializada)

- solicita extração dos conceitos requisitados na consulta
- percorre a árvore retornada do processo de extração (pega o 1º nó)

{

- insere informação nas tabelas de relacionamento entre elementos não léxicos
- se o identificador do objeto já está na tabela do conceito
  - compara valores de todos os campos para este objeto
    - se são iguais: pega próximo nó
    - se são diferentes: chama a rotina **X**

- se o identificador do objeto não está na tabela do conceito
  - insere este objeto
  - pega próximo nó

}

- atualiza as tabelas *Conceitos_materializados* e *Documentos*

⇒ se o rótulo temporal na tabela Documentos for mais antigo que a data de última atualização retornada da fonte XML (significa que a visão materializada está desatualizada em relação às fontes de dados)

- solicita extração dos conceitos requisitados na consulta e dos conceitos que já estão materializados para essa fonte de dados (tabela Conceitos_materializados)

-percorre a árvore retornada do processo de extração (pega o 1º nó)

{

- insere, se necessário, informação nas tabelas de relacionamento entre elementos não léxicos
- se o identificador do objeto já está na tabela do conceito
  - compara valores de todos os campos para este objeto
    - se são iguais: pega próximo nó
    - se são diferentes: chama a rotina **X**
- se o identificador do objeto não está na tabela do conceito
  - insere este objeto
  - pega próximo nó

}

-após percorrer toda a árvore, verifica se existe algum identificador de objeto que está nas tabelas de relacionamento para essa fonte, mas não na árvore recebida.

-Se sim:

-remove das tabelas de relacionamento

-remove o id_conceito da tabela *Conceitos_materializados* para esta fonte

-verifica se na tabela *Conceitos_materializados* alguma outra fonte referencia esse identificador:

-se não:

-remove a informação da tabela do conceito

-atualiza tabelas *Conceitos_materializados* e *Documentos*

⇒ se rótulo temporal na tabela Documentos for igual à data de última atualização retornada da fonte XML (significa que a visão materializada está atualizada em relação às fontes de dados)

- verificar se a visão já possui materializado o conceito requisitado na consulta (ver na tabela *Conceitos_materializados)*
  - se sim: pode responder a consulta com o conteúdo da visão materializada

- se não:
  - solicita extração:
    - conceitos solicitados na consulta e que não estão na visão materializada

  - percorre a árvore retornada do processo de extração (pega o 1º nó)
  {
  - insere informação nas tabelas de relacionamento entre elementos léxicos não léxicos
  - se o identificador do objeto já está na tabela do conceito
    - compara valores de todos os campos para este objeto
      - se são iguais: pega próximo nó
      - se são diferentes: chama a rotina X
    - se o identificador do objeto não está na tabela do conceito
      - insere este objeto
      - pega próximo nó

}

- atualiza tabela *Conceitos_materializados*

Rotina X: essa rotina é utilizada sempre que o valor de um campo apresenta valores conflitantes, ou seja, o valor retornado da extração para uma determinada propriedade é diferente do valor armazenado na visão materializada. Neste caso, utiliza-se a informação da fonte que tiver o rótulo temporal mais atual. Para verificar qual fonte possui o rótulo temporal mais atual, o seguinte algoritmo é proposto para esta rotina:

- procurar na tabela *Conceitos_materializados*, para o identificador do objeto em questão, quais as outras fontes de dados que referenciam este objeto
- se nenhuma outra fonte o referencia, vai para o próximo nó (o valor desta propriedade não é atualizado na visão materializada)
- se alguma outra fonte de dados o referencia, consultar a tabela *Documentos* e verificar qual delas tem o rotulo temporal mais atual;
- se esse rótulo temporal for mais atual que o da fonte de dados recém extraída: não atualiza o campo na tabela do conceito
- se esse rótulo temporal não for mais atual que o da fonte de dados recém extraída: atualiza a tabela do conceito

A seguir, são mostrados três exemplos de consultas do usuário. Primeiramente, suponha que o banco de dados esteja vazio. Assim, qualquer consulta do usuário implica em uma extração de dados XML para posterior materialização.

Primeiramente, considere o caso em que são solicitados *nomes de autores*. Suponha que o documento XML retornado da fonte de dados seja a árvore representada na FIGURA 11. A inserção dos dados no banco de dados dá-se da seguinte maneira:

TABELA 26 - Tabela Autor após a Inserção dos Dados Referentes à FIGURA 11

Id_autor	**Nome**
MARCOSSANTOS	Marcos Alberto Santos

TABELA 27 - Tabela Documentos após a Inserção dos Dados Referentes à FIGURA 11

Id_fonte	**Rotulo_temporal**
www.inf.ufrgs.br/~santos/doc1.xml	21/07/2000

TABELA 28 - Tabela Conceitos Materializados após a Inserção dos Dados Referentes à FIGURA 11

Id_fonte	**Conceito_ontologia**	**Id_instancia**
www.inf.ufrgs.br/~santos/doc1.xml	autor	MARCOSSANTOS
www.inf.ufrgs.br/~santos/doc1.xml	nome	Marcos Alberto Santos

Apenas inserção de registros foi realizada, de acordo com o *caso 1,* apresentado anteriormente.

Considere agora um outro exemplo. Uma segunda consulta do usuário também solicita *nomes de autores.* Primeiramente, é verificado se os dados já materializados estão consistentes em relação às fontes. Para isso, é solicitada a data de última alteração da fonte de dados identificada por *www.inf.ufrgs.br/~santos/doc1.xml*. Supondo que a data retornada desta fonte seja *02/09/2000.* Isso implica que os dados materializados estão desatualizados, pois o rótulo temporal desta fonte, na tabela *documentos*, é *21/07/2000.* Desse modo, uma nova extração dos conceitos solicitados na consulta é realizada. De posse dos identificadores das instâncias dos dados XML, representados na árvore por nós ID, pode-se fazer a manutenção do banco de dados.

Considerando que o processo de extração retorne a árvore da FIGURA 12:

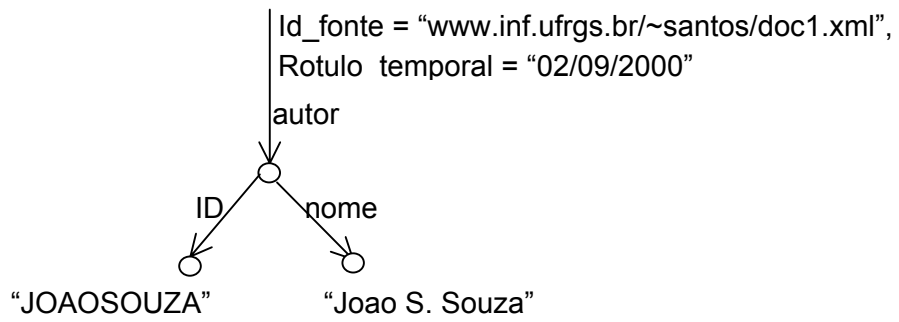

FIGURA 12 - Árvore representativa do documento XML, retornada do processo de extração

Utilizando a técnica anteriormente apresentada para a manutenção do banco de dados, ter-se-ia:

TABELA 29 - Tabela Autor após a Inserção dos Dados Referentes à FIGURA 12

Id_autor	**Nome**
JOAOSOUZA	Joao S. Souza

TABELA 30 - Tabela Documentos após a Inserção dos Dados Referentes à FIGURA 12

Id_fonte	**Rotulo_temporal**
www.inf.ufrgs.br/~santos/doc1.xml	02/09/2000

TABELA 31 - Tabela Conceitos Materializados após a Inserção dos Dados Referentes à FIGURA 12

Id_fonte	**Conceito_ontologia**	**Id_instancia**
www.inf.ufrgs.br/~santos/doc1.xml	autor	JOAOSOUZA
www.inf.ufrgs.br/~santos/doc1.xml	nome	Joao S. Souza

Nota-se que foram realizadas remoções (*caso 2*) e inserções (*caso 1*) de registros das tabelas do banco de dados.

Finalmente, considere uma terceira consulta do usuário, solicitando também *nomes de autores*. Suponha que uma outra fonte de dados XML tenha ingressado no sistema que faz a materialização das fontes participantes. Consequentemente, no momento da consulta, uma extração de dados deve ser solicitada à esta fonte. Considere que a árvore retornada do processo de extração, para esta fonte, esteja representada na FIGURA 13.

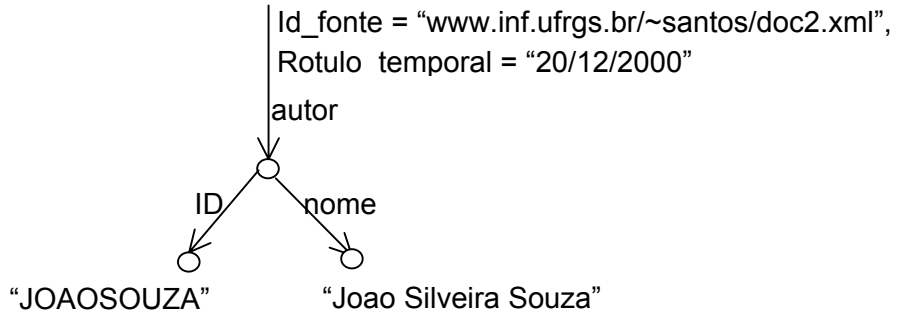

FIGURA 13 - Árvore representativa do documento XML, retornada do processo de extração

Após a manutenção do banco de dados, ter-se-ia o seguinte:

TABELA 32 - Tabela Autor após a após a Inserção dos Dados Referentes à FIGURA 13

Id_autor	Nome
JOAOSOUZA	Joao Silveira Souza

TABELA 33 - Tabela Documentos após a Inserção dos Dados Referentes à FIGURA 13

Id_fonte	Rotulo_temporal
www.inf.ufrgs.br/~santos/doc1.xml	02/09/2000
www.inf.ufrgs.br/doc2.xml	20/12/2000

TABELA 34 - Tabela Conceitos Materializados após a Inserção dos Dados Referentes à FIGURA 13

Id_fonte	Conceito_ontologia	Id_instancia
www.inf.ufrgs.br/~santos/doc1.xml	autor	JOAOSOUZA
www.inf.ufrgs.br/~santos/doc1.xml	nome	Joao Silveira Souza
www.inf.ufrgs.br/doc2.xml	autor	JOAOSOUZA
www.inf.ufrgs.br/doc2.xml	nome	Joao Silveira Souza

A existência de um conflito no valor da propriedade *nome* para um mesmo autor, ocasionou uma atualização no registro da tabela *autor (caso 3).* Note que o rótulo temporal da árvore extraída é mais atual do que o rótulo temporal da fonte que anteriormente tinha materializado o valor desta propriedade para o mesmo autor.

4.5.3 Resumo

Diversos pontos ainda continuam em aberto na área de dados semi-estruturados, dada a carência de literatura a respeito, como por exemplo, atualização de dados e propagação de modificações. Particularmente, os problemas de atualização e interoperabilidade são complexos, pois dados semi-estruturados evoluem rapidamente e são heterogêneos.

As técnicas encontradas na literatura para manutenção incremental de visões XML baseiam-se fortemente na existência de identificadores de objetos. Sistemas específicos propõem algumas soluções para este problema [McH97, LIE99]. Quando não existem esses identificadores, a manutenção torna-se mais difícil, haja visto a dificuldade em se identificar as instancias que devem ser mantidas na visão materializada.

Neste sentido, nota-se a grande importância da existência de um mecanismo de identificação de instâncias XML para a adoção de uma técnica para a manutenção da visão materializada. Com a proposta de identificação de instâncias deste trabalho (seção 4.4), este problema foi contornado. Partindo-se do pressuposto que cada documento XML possui um rótulo temporal que identifica a sua data de última atualização, foi proposta uma técnica para manutenção incremental da visão materializada definida.

5 Conclusões e Trabalhos Futuros

A proposta deste trabalho apresenta uma técnica para materialização relacional de dados XML. As principais contribuições são:

- propõe uma técnica de geração da estrutura relacional do banco de dados, baseada no uso de uma ontologia que descreve o domínio do problema das fontes de dados. Com esta proposta, o conhecimento semântico que é capturado durante a transformação de modelos assegura a geração de um esquema relacional correto e flexível. Tal conhecimento semântico pode ser adquirido a partir da ontologia. A ontologia é utilizada como o esquema conceitual do sistema integrado, independente de implementação. Desta maneira, várias DTDs diferentes podem estar de acordo com a ontologia descrita. Uma vez que uma única DTD pode validar vários documentos XML, pode-se verificar que uma variedade maior de documentos XML podem ser inseridos no banco de dados relacional, já que o esquema é gerado a partir da ontologia;

- apresenta o problema de identificação de instâncias XML, propondo um mecanismo de atribuição de identificadores a instâncias de dados XML, baseado em funções *Skolem* e no padrão *XPath*. Este é o principal foco do trabalho. Uma vez que a materialização das visões XML é feita em um BD relacional, é de fundamental importância realizar a identificação das instâncias. O estado da arte mostra algumas propostas de atribuição de identificadores à instâncias XML, mas estas propostas apresentam-se restritas quando trabalha-se com uma abordagem OWA, como a utilizada neste trabalho. Neste sentido, a técnica proposta apresenta-se como uma boa alternativa para a identificação de instâncias XML;

- propõe uma técnica de manutenção incremental do BD relacional, a fim de mantê-lo consistente em relação às fontes de dados, à medida que estas sofrem alterações. O mecanismo de atualização incremental apresentado é relativamente simples, mas atinge os objetos propostos. Parte-se do pressuposto que cada fonte de dados XML tem como identificar sua data de última atualização. A partir deste rótulo temporal e da técnica proposta para identificação das instâncias XML, o algoritmo apresentado realiza a manutenção da visão. No entanto, rótulos temporais não estão obrigatoriamente presentes em fontes da WEB, o que caracteriza um ponto fraco da técnica proposta. Outra deficiência a ser apontada é a necessidade de se conhecer a sintaxe *XPath*.

Apesar dos obstáculos em converter-se XML para modelos relacionais, diversos benefícios podem ser apontados: considerando o mercado atual que é dominado na maior parte por produtos relacionais, não é fácil nem prático abandonar esta tecnologia e migrar para tecnologias que suportem XML. Adotar um BD relacional como um sistema de armazenamento para dados XML implica em poder utilizar várias técnicas já bastante amadurecidas desta tecnologia, como por exemplo: OLAP, mineração dos dados, otimização de consultas, SQL, manutenção de visões, etc. Além disso, a integração de dados XML com os dados legados em formato relacional é possível.

Na forma em que se encontra o presente trabalho, muito trabalho manual é exigido do administrador, principalmente em relação à identificação das instâncias, pois é exigido do

usuário o conhecimento e a utilização de *XPath*. Para tornar a proposta utilizável por usuários que não dominam estas técnicas, é necessária a construção de uma interface gráfica na qual se possa escolher entre esqueletos pré-construídos de consultas *XPath*.

O pressuposto da existência de um rótulo temporal nas fontes semi-estruturadas facilita bastante o processo de manutenção. No entanto, nem sempre esta informação é disponível, como no caso das fontes WEB. Para isso, um mecanismo mais sofisticado de manutenção incremental de visões materializadas pode ser proposto, sem fazer uso de rótulos temporais;

O problema de correspondência entre os elementos do esquema integrado e os elementos dos esquemas locais foi solucionado através da inclusão de uma tabela *sinônimos*. Uma técnica mais sofisticada, utilizando, por exemplo, bases de conhecimento de conceitos e termos ou mesmo a ontologia poderia ser utilizada para estabelecer esta correspondência.

A técnica proposta para o problema da identificação de instâncias baseou-se em características inerentes, próprias dos objetos XML. Um trabalho futuro pode utilizar a ontologia para esta finalidade, incluindo a informação necessária para a identificação de instâncias.

A tendência é a popularização cada vez mais rápida de dados XML. Deste modo, torna-se de grande importância a busca por novas técnicas eficientes de integração, armazenamento e consultas a estes dados.

Bibliografia

[ABI 97] ABITEBOUL, S. et al. Views for Semistructured Data. In: WORKSHOP ON MANAGEMENT OF SEMISTRUCTURED DATA, 1997. Disponível em: <citeseer.nj.nec.com/article/abiteboul77views.html>. Acesso em: 20 mar. 2000.

[ABI 97a] ABITEBOUL, S. et al. Querying Semi-Structured Data. In: INTERNATIONAL CONFERENCE ON DATABASE THEORY, ICDT, 6., 1997, Delphi, GR. **Database Theory:** proceedings. Berlin: Springer-Verlag, 1997. Disponível em: <http://citeseer.nj.nec.com/abiteboul97querying.html>. Acesso em: 22 abr. 2000.

[ABI 98] ABITEBOUL, S. et al. Incremental Maintenance for Materialized Views over Semistructured Data. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, VLDB, 24., 1998, New York. **Proceedings...** [S.l.:s.n.], 1999. p. 36-49.

[AHM91] AHMED, R. et al. The Pegasus Heterogeneous Multidatabase System. **Computer**, New York, v.24, n.12, p. 19-27, Dec. 1991.

[ALB96] ALBERT, J. Data Integration in the RODIN Multidatabase System. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, 1., 1996. **Papers**. [S.l.:s.n.], 1996. p. 48-57.

[ALI 2000] ALI, M. A. et al. Incremental Maintenance of Materialized OQL Views. In: INTERNATIONAL WORKSHOP ON DATA WAREHOUSING AND OLAP, DOLAP, 3., 2000, Washington. **Proceedings...** [S.l.:s.n.], 1998.

[BOU 2001] BOURRET, R. **Mapping DTDs to Databases**. 2001. Disponível em: <http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html>. Acesso em: 26 set. 2001.

[BOU 2001a] BOURRET, R. **XML and Databases**. Disponível em: <http://www.rpbourret.com/xml/XMLAndDatabases.htm>. Acesso em: 29 set. 2001.

[BOU 2001b] BOURRET, R. **XPath in Five Paragraphs**. Disponível em: <http://www.rpbourret.com/xml/XPathIn5.htm>. Acesso em: 30 set. 2001.

[BRA 2000] BRADLEY, Neil. **The XML Companion**. 2nd ed. Harlow: Addison-Wesley, 2000.

[CHA 91] CHATTERJEE, A. et al. Data Manipulation in Heterogeneous Databases. **SIGMOD Record**, New York, v.2, n.4, p. 64-68, Dec. 1991.

[CHO 2000] CHO, J. et al. Finding replicated Web collections. **SIGMOD Record**, New York, v.29, n.2, p. 355-366, June 2000. Trabalho apresentado na ACM

SIGMOD International Conference on Management of Data, SIGMOD, 18., 2000, Dallas.

[CHR 94] CHRISTOPHIDES, V. et al. From structured documents to novel query facilities. **SIGMOD Record**, New York, v.23, n.2, p. 313-324, June. 1994.

[DAY 83] DAYAL, U. Processing queries over generalized hierarchies in a multidatabase system. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 9., 1983, Florence, IT. **Proceedings...** Florence: VLDB Endowment, 1983. p. 342-353.

[DeM 89] DEMICHIEL, L. G. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains**. IEEE Transactions on Knowledge and Data Engineering**, New York, v.1, n.2, p. 485-493, Dec. 1989.

[DEU 99] DEUTSCH, A. et al. Storing Semistructured Data in Relations. In: WORKSHOP ON QUERY LANGUAGES FOR SEMISTRUCTURED DATA AND NON-STANDARD DATA FORMATS, 1999, Jerusalem, IS. **Proceedings...** Jerusalem: [s.n.], 1999.

[DEU 99a] DEUTSCH, A. et al. Storing Semistructured Data with STORED. **SIGMOD Record,** New York, p.431-442, June 1999. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, SIGMOD, 1999, Philadelphia.

[DEU 99b] DEUTSCH, A. et al. A query language for XML**. Journal WWW8/ Computer Networks**, [S.l.], v.31, n.11-16, p.1155-1169. Disponível em: <http://www.research.att.com/~mff/files/final.html>. Acesso em: 24 abr. 2001.

[DOR 2000] DORNELES, C. F. **Extração de Dados Semi-Estruturados com base em uma Ontologia**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[EMB 98] EMBLEY, D. et al. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 1998, Bethesda, US. **Proceedings...** Bethesda: [s.n.], 1998. p. 52-59.

[EMB 98a] EMBLEY, D. et al. A Conceptual-Modeling Approach to Extracting Data from the Web. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELLING, 17., 1998, Singapore. **Proceedings...** Singapore: [s.n.], 1998. p. 78-91.

[ERD 2001] ERDMANN, M. et al. How to Structure and Access XML Documents with Ontologies. **Data & Knowledge Engineering,** [S.l.], v.36, n.3, Mar. 2001.

[FER 2000] FERNANDEZ, M. F. et al. SilkRoute: trading between relations and XML. **Computer Networks**, [S.l.], v.33, n.1-6, p. 723-745, June 2000. Trabalho apresentado na International World Wide Web Conference, 9., 2000.

[FER 98] FERNANDEZ, M. et al. Catching the Boat with Strudel: Experiences with a Web-Site Management System. **SIGMOD Record**, New York, v.27, n.2, p. 414, 1998. Trabalho apresentado na ACM SIGMOD Special Interest Group on Management of Data, SIGMOD, 1998.

[FLO 99] FLORESCU, D. et al. **A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database.** 1999. (Technical Report, n. 3680). Disponível em: <http://rodin.inria.fr/Epubsbyyear.html>. Acesso em: 09 maio 2001.

[FLO 99a] FLORESCU, D. et al. Storing and Querying XML Data using an RDMBS**. IEEE Data Engineering Bulletin**, New York, v.22, n.3, p. 27-34, Sept. 1999.

[GOG 95] GOGOLLA, M. Identifying Objects by Declarative Queries. In: CHOMICKI, Jan; SAAKE, Gunter; SERNADAS, Christina**. The Role of Logics in Information Systems**. [S.l.:s.n.], 1995. (Dagstuhl-Seminar-Report, n. 121).

[GUP 92] GUPTA, A. et al. **Maintenance Policies. Materialized Views–Techniques, Implementations and Applications.** [S.l.]: Massachusetts Institute of Technology, 1999.

[GUP 99] GUPTA, A. et al. **Materialized Views–Techniques, Implementations and Applications**. [S.l.]: Massachusetts Institute of Technology, 1999.

[HEI 95] HEIN, J. **Discrete Structures, Logic and Computability**. [S.l.]: Jones&Bartlett Publishers, 1995. Preliminary Edition.

[HUL 90] HULL, R. et al. ILOG: Declarative Creation and Manipulation of Object Identifiers. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 6., 1990, Brisbane, AU. **Proceedings...** Brisbane: VLDB Endowment, 1990. p. 455-468.

[KAN 2000] KANTORSKI, G. Z. et al. Heterogeneous Database Interoperability Using the WWW. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 15., 2000, João Pessoa, BR. **Proceedings...** João Pessoa: [s.n.], 2000. p.79-88.

[KAY 2000] KAY, M. **XSLT Programmer´s Reference**. USA: Wrox Press, 2000.

[LIE 99] LIEFKE, H. et al. **Efficient View Maintenance in XML Data Warehouses**. [S.l.]: Department of Computer and Information Science, University of Pennsylvania. Disponível em: <http://www.cis.upenn.edu/~liefke/papers/whax.ps.gz>. Acesso em: 27 abr. 2001.

[LIM 93] LIM, E. et al. Entity identification in database integration. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 9., 1993, Viena, AU. **Proceedings...** Viena: [s.n.], 1993. p.294-301.

[LIM 94] LIM, E. et al. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 10., 1994, Houston, US. **Proceedings...** Houston: [s.n.], 1994. p.154-163.

[LIM 98] LIM, E. et al. A Global Object Model for Accommodating Instance Heterogeneities. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELLING, 17., 1998, Singapore. **Proceedings...** Singapore: [s.n], 1998. p. 435-448.

[MAG 2001] MAGALHÃES, K. et al. Uma Abordagem para Armazenamento de Dados Semi-Estruturados em Bancos de Dados Relacionais. Artigo submetido ao XVI Simpósio Brasileiro de Banco de Dados, Rio de Janeiro, outubro 2001.

[MAN 2000] MANOLESCU, I. et al. Agora: Living with XML and Relational. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 26., 2000, Cairo, EG. **Proceedings...** Cairo: VLDB Endowment, 2000. p. 623-626

[MCH97] McHUGH, J. et al. Lore: A Database Management System for Semistructured Data. Disponível em: <http://www-db.stanford.edu/lore/pubs/lore97.pdf>. **SIGMOD Record**, New York, v.26, n.3, p. 54-66, Sept. 1997. Acesso em: 28 jun. 2001.

[MEL 2000] MELLO, R. S. et al. Dados Semi-Estruturados. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 15., 2000, João Pessoa. **Anais...** João Pessoa: [s.n.], 2000.

[MEL 2000a] MELLO, R. S. **Aplicação de Ontologias a Bancos de Dados Semi-Estruturados**. 2000. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MOT 81] MOTRO, A. et al. Constructing Superviews. **SIGMOD Record,** New York, p.56-64, May 1981. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, SIGMOD, 1981, Michigan.

[NEV 2000] NEVES, A. P. **Lógica de Primeira Ordem**. Disponível em: <http://www.est.ipcb.pt/disciplinas/LPO/Teorica%2011.pdf>. Acesso em: 19 jul. 2001.

[PAP 95] PAPAKONSTANTINOU, Y. et al. Object Exchange Across Heterogeneous Information Sources. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 11., 1995, Taipei, TA. **Proceedings...** Taipei: [s.n.], 1995. p. 251-260.

[PAP 96] PAPAKONSTANTINOU, Y. et al. Object Fusion in Mediator Systems. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 22., 1996, Bombay. **Proceedings...** Bombay: VLDB Endowment, 1996. p. 413-424.

[RED 94] REDDY, M. P. et al. A Methodology for Integration of Heterogeneous Databases. **IEEE Transactions on Knowledge and Data Engineering**, New York, v.6, n.6, p. 920-933, Dec. 1994.

[SAL 2001] SALGADO, A . C. et al. Integração de Dados na WEB. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 21., 2001, Fortaleza. **Anais...** Fortaleza: [s.n.], 2001.

[SHA 99] SHANMUGASUNDARAM, J. et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 25., 1999, Edinburg, UK. **Proceedings...** Edinburg: VLDB Endowment, 1999.

[SHO 93] SHOENS, K. A. et al. The Rufus System: Information Organization for Semi-Structured Data. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 19., 1993, Dublin, IR. **Proceedings...** Dublin: VLDB Endowment, 1993. p. 97-107.

[SIL 2000] SILVA, A. S. **Materialização de Visões Relacionais para Dados Semi-Estruturados através de Ontologias**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[SIL 2000a] SILVA, A. S. et al. Materialização de Visões para dados XML através de Ontologias. In: WORKSHOP DO PROJETO IDOC, WIDOC, 2000, Curitiba. **Anais...** Curitiba: [s.n.], 2000.

[SUC96] SUCIU, D. Query Decomposition and View Maintenance for Query Languages for Unstructured Data. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 22., 1996, Mumbai, IN. **Proceedings...** Mumbai: VLDB Endowment, 1996. p. 227-238.

[TSE 92] TSENG, F. S. et al. A Probabilistic Approach to Query Processing in Heterogeneous Database Systems. In: INTERNATIONAL WORKSHOP ON RESEARCH ISSUES ON DATA ENGINEERING: TRANSACTION AND QUERY PROCESSING, 2., 1992, Tempe, US. **Proceedings...** Tempe: [s.n.], 1992. p. 176-183.

[W3C 99] WORLD WIDE WEB CONSORTIUM**. XML Path Language (XPath) Version 1.0**-W3C Recommendation 16 November 1999. Disponível em: <http://www.w3.org/TR/xpath>. Acesso em: 20 abr. 2001.

[WAN 89] WANG, Y. R. et al. The Inter-Database Instance Identification Problem in Integrating Autonomous Systems. In: INTERNATIONAL CONFERENCE ON

DATA ENGINEERING, 5., 1989, Los Angeles, US. **Proceedings...** Los Angeles: [s.n.], 1989. p. 46-55.

[WIE 97] WIENER, J. L. et al. The WHIPS Prototype for Data Warehouse Creation and Maintenance. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 13., 1997, Birmingham, 1997, UK. **Proceedings...** Birmingham: [s.n.], 1997.

[ZHO 95] ZHOU, G. et al. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In: INTERNATIONAL CONFERENCE ON COOPERATIVE INFORMATION SYSTEMS, 3., 1995, Viena, AU. **Proceedings...** Viena: [s.n.], 1995. p. 4-18.

[ZHU 98] ZHUGE, Y. et al. Graph Structured Views and Their Incremental Maintenance. In: CONFERENCE ON DATA ENGINEERING, 14., 1998, Orlando, US. **Proceedings...** Orlando: [s.n.], 1998. p. 116-125.

[ZWO 99] ZWOL, R. et al. Modelling and querying semistructured data with MOA. In: WORKSHOP ON SEMI-STRUCTURED DATA AND NONSTANDARD DATA FORMATS, 1999, Jerusalem, IS. **Proceedings...** Jerusalem: VLDB Endowment, 1999.