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

FELIPE VICTOLLA SILVEIRA

Fragmentação e Decomposição de Consultas em XML

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof.Dr. Carlos A. Heuser
Orientador

Porto Alegre, novembro de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silveira, Felipe Victolla

Fragmentação e Decomposição de Consultas em XML / Felipe Victolla Silveira. – Porto Alegre: PPGC da UFRGS, 2006.

51 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientador: Carlos A. Heuser.

1. Fragmentação de bases de dados. 2. Decomposição de consultas. 3. XML I. Heuser, Carlos A.. II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

XPath	XML Path
XML	Extensible Markup Language
CXPath	Conceptual XPath
TAX	Tree Algebra for XML
BInXS	Bottom-up Integration of XML Schematas
OQL	Object Query Language
CQuery	Concept Query
LAV	Local-as-view
GAV	Global-as-view

LISTA DE FIGURAS

RESUMO

O problema da integração de dados (fragmentação de dados, decomposição de consultas) tem sido largamente estudado na literatura, mas a estrutura hierárquica inerente do modelo XML apresenta problemas que são específicos deste modelo de dados. Cada relacionamento conceitual muitos-para-muitos deve ser mapeado para uma estrutura hierárquica específica em XML. Diferentes fontes XML podem implementar o mesmo relacionamento conceitual muitos-para-muitos de diferentes maneiras. Na abordagem proposta neste trabalho, o problema de integração de fontes de dados XML é dividido em dois problemas: (1) naquele da fragmentação de um modelo global do tipo grafo (ex., um modelo ER) em diversos modelos locais do tipo grafo representando conceitualmente fontes de dados e (2) naquele do mapeamento de um modelo local do tipo grafo em um esquema hierárquico XML. Este trabalho apresenta um conjunto de operadores especificamente projetados para esta abordagem, assim como um mecanismo de decomposição que permite que uma consulta especificada em um nível conceitual seja decomposta em uma consulta XQuery especificada no nível XML. Como linguagem de consulta para o nível conceitual, é adotado o CXPath (Conceptual XPath), uma linguagem de consulta proposta em um trabalho anterior.

Palavras-chave: Fragmentação de bases de dados, Decomposição de consultas, XML.

Fragmentation and Query Decomposition in XML

ABSTRACT

The problem of data integration (query decomposition, data fragmentation) has been widely studied in literature, but the inherent hierarchical nature of XML data presents problems that are specific to this data model. Each many-to-many conceptual relationship must be mapped to a specific hierarchical structure in XML. Different XML sources may implement the same many-to-many conceptual relationship in different ways. In our approach the problem of integration of XML data sources is decomposed in two problems: (1) that of fragmentation of a global graph-like model (e.g., an ER model) into several local graph-like models conceptually representing data sources and (2) that of mapping the local graph-like model into an XML tree-like schema. This work presents a set of fragmentation operators specifically designed for our approach, as well as a query decomposition mechanism that allows a query stated at the conceptual level to be decomposed into an XQuery statement at the XML level. As the query language at the conceptual level, we adopt CXPath (conceptual XPath) a query language we have defined in previous work.

Keywords: Database fragmentation, Query Decomposition, XML.

1 INTRODUÇÃO

Neste trabalho, é tratado o problema de consultar várias fontes XML que possivelmente possuem diferentes esquemas e pertencem a um domínio comum. É aplicada a abordagem de mediadores (WIEDERHOLD, 1992), na qual as consultas são submetidas contra um esquema global ou mediado. Neste contexto, uma importante decisão é escolher o nível de abstração do modelo de dados usado no nível global. Uma abordagem adotada por alguns autores é utilizar o mesmo nível de abstração nos níveis global e local (DOAN; DOMINGOS; HALEVY, 2001; MA et al., 2003; MA; SCHEWE, 2003; REYNAUD; SIROT; VODISLAV, 2001). Na integração de fontes XML isso significa que o esquema global é um esquema XML. Esta abordagem tem a potencial vantagem de simplificar a tradução de consultas do modelo global para um esquema local, mas também apresenta um problema maior: devido a estrutura hierárquica inerente dos documentos XML, construções conceituais não-hierárquicas como relacionamentos muitos-para-muitos precisam ser hierarquicamente representados no esquema XML.

Em um trabalho anterior foi adotada outra abordagem, especificamente aquela de usar um modelo de dados mais abstrato no nível global. Em (MELLO; CASTANO; HEUSER, 2002; MELLO; HEUSER, 2001, 2005) foi proposto o uso de um modelo conceitual do tipo entidade-relacionamento para os níveis global e local, e é mostrado como cada esquema XML local é abstraído para um esquema conceitual local, e como estes esquemas conceituais locais são integrados em um esquema conceitual global. Consultas contra o esquema conceitual são especificadas em CXPath (conceptual XPath), uma linguagem de consulta baseada no XPath, e traduzidas em uma consulta XPath através de uma abordagem de reescrita de consultas (CAMILLO; MELLO; HEUSER, 2003).

Entretanto, o mecanismo proposto em (CAMILLO; MELLO; HEUSER, 2003) é limitado à tradução de uma fonte de cada vez. As consultas locais geradas por este mecanismo são especificadas contra uma única fonte, limitando esta abordagem a consultas que não exijam interação entre múltiplas fontes de dados para a construção da resposta. Neste trabalho esta limitação é tratada, que é como decompor uma consulta global especificada contra um esquema global em consultas locais especificadas contra diversas fontes XML. Dado este contexto, existem dois problemas principais que precisam ser endereçados. O primeiro problema consiste em definir como um esquema conceitual global é fragmentado em diversos esquemas locais. O segundo problema consiste em definir um algoritmo, baseado na fragmentação dos esquemas, que irá decompor uma consulta global em diversas consultas locais.

Neste trabalho, é proposta uma solução para ambos os problemas. Primeiro, operadores de fragmentação para dados XML são introduzidos, considerando a ex-

istência de um esquema conceitual global. São definidos três operadores de fragmentação contra um esquema conceitual e uma base conceitual: fragmentação *split*, fragmentação *vertical* e fragmentação *horizontal*. Adicionalmente, é proposto um algoritmo para decomposição de consultas, baseado no uso de informações de mapeamento entre os esquemas conceituais locais e os documentos XML. Este algoritmo gera expressões XPath que acessam as fontes individuais, e a seguir integram estas expressões em uma única consulta XQuery (W3C, 1998), tratando a fragmentação das fontes e construindo a resposta para a consulta. O comportamento deste algoritmo é baseado em como as fontes estão fragmentadas. Isto significa que são implementados algoritmos para redução da fragmentação, da mesma forma como é feito em bancos de dados distribuídos (OZSU; VALDURIEZ, 1991).

O problema de consultar fontes de dados integradas tem sido investigado por diversos autores (JOSIFOVSKI; RISCH, 2002; MACKINNON; MARWICK; WILLIAMS, 1998; MANOLESCU; FLORESCU; KOSSMANN, 2001; SATTLER; GEIST; SCHALLEHN, 2005; SUCIU, 2002). Uma abordagem similar à apresentada neste trabalho é a descrita em (MACKINNON; MARWICK; WILLIAMS, 1998). Nesta abordagem, as consultas também são expressas em um nível conceitual. As consultas são escritas em uma interface restrita, ligada a uma ontologia de domínio, e em seguida decomposta para as bases individuais. A diferença para a abordagem proposta neste trabalho é que as fontes não são documentos XML, mas sim bases relacionais. Uma abordagem que integra fontes de dados XML é uma proposta em (MANOLESCU; FLORESCU; KOSSMANN, 2001). Nesta abordagem, as consultas são especificadas contra um esquema global hierárquico (modelo XML), ao invés de um esquema global do tipo grafo como o proposto neste trabalho. Em (JOSIFOVSKI; RISCH, 2002), o esquema mediado é do tipo orientado a objetos, e a linguagem de consulta definida para consultar este modelo é uma variação do OQL. A abordagem que mais se aproxima daquela proposta neste trabalho é o sistema mediador Yacob (SATTLER; GEIST; SCHALLEHN, 2005). No Yacob, as consultas escritas contra várias fontes relacionadas a um domínio comum são especificadas contra um modelo de integração baseado em conceitos, similar ao RDF, e a seguir traduzidas para as fontes individuais. As consultas são expressas em CQuery, uma linguagem utilizada para expressar consultas contra o modelo ontológico. CQuery é baseada no XQuery, enquanto que a linguagem utilizada neste trabalho é baseada no XPath. Outra diferença entre o Yacob e a abordagem proposta neste trabalho é a de que a segunda se baseia em um conjunto de operadores de fragmentação especificamente projetados para decompor um esquema conceitual em um esquema XML.

Este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta o estado da arte em fragmentação e decomposição de consultas em bases distribuídas. O Capítulo 3 apresenta o modelo de dados utilizado pela abordagem proposta neste trabalho. O Capítulo 4 descreve os operadores de fragmentação. O Capítulo 5 descreve as informações de mapeamento e o Capítulo 6 descreve o algoritmo de decomposição e sua aplicação através de um exemplo. O Capítulo 6 é dedicado às conclusões e trabalhos futuros.

2 CONSULTAS EM FONTES DISTRIBUÍDAS

Este Capítulo apresenta o estado da arte em consultas sobre fontes distribuídas, estando organizado em quatro Seções. Na Seção 2.1, são discutidas as abordagens para fragmentação em fontes distribuídas. Na Seção 2.2, são apresentadas abordagens para decomposição de consultas em bases de dados distribuídas e XML. Na Seção 2.3, é apresentada a abordagem para integração de esquemas XML BInXS e a linguagem de consultas CXPath, desenvolvidas por nosso grupo de pesquisa. Por fim, na Seção 2.4, são apresentadas as conclusões.

2.1 Fragmentação

Em um ambiente de banco de dados distribuído, a base de dados precisa ser dividida em duas ou mais partes de tal forma que a combinação dessas partes leve a base de dados global sem qualquer perda de informação. Cada parte resultante é conhecida como fragmento de dados (TAMHANKAR; RAM, 1998). A meta da fragmentação é melhorar a performance e aumentar a disponibilidade dos dados em um ambiente distribuído (MA et al., 2003).

A seguir, são discutidos alguns dos trabalhos existentes na literatura sobre fragmentação de dados em bases distribuídas. Esta seção está dividida em duas partes. Na primeira parte, serão apresentados os conceitos para fragmentação em bases de dados relacionais. Na segunda parte, serão apresentados trabalhos que tratam de fragmentação de dados em fontes XML.

2.1.1 Fragmentação em Bases Relacionais Distribuídas

No contexto de bases relacionais distribuídas, existem basicamente duas abordagens para fragmentação: horizontal e vertical. Um fragmento horizontal consiste em um subconjunto das tuplas de uma determinada relação (CERI; NEGRI; PELAGATTI, 1982). Ele é definido através de uma operação de seleção da álgebra relacional sobre uma determinada relação. Existem dois tipos de fragmentação horizontal: primária e derivada. A fragmentação horizontal primária consiste na aplicação de um determinado predicado de seleção sobre a relação. Já a fragmentação horizontal derivada é o particionamento de uma relação que resulta da aplicação de predicados sobre outra relação.

Um fragmento vertical consiste em um subconjunto de atributos de uma determinada relação (NAVATHE et al., 1984). Ele é definido através de uma operação de projeção, através da qual é selecionado apenas um subconjunto de atributos, mais a chave primária da relação. O objetivo de incluir a chave primária da relação em

todos os seus fragmentos verticais é possibilitar a reconstrução da relação global através de uma junção dos fragmentos.

Em (CERI; PELAGATTI, 1983), são definidas regras de correção para as operações de fragmentação. Essas regras garantem que o banco de dados não sofra nenhuma mudança semântica durante o processo de fragmentação (OZSU; VALDURIEZ, 1991). As regras são as seguintes: *completeza*, *reconstrução* e *disjunção*. Completeza significa que qualquer instância que pode ser encontrada em uma relação também pode ser encontrada em algum de seus fragmentos. Reconstrução significa que existe uma operação capaz de reconstruir a relação original a partir dos fragmentos. Disjunção significa que, se uma dada instância de uma relação está em um dos fragmentos, ela não está em nenhum dos outros fragmentos.

2.1.2 Fragmentação em XML

A maior parte dos trabalhos existentes na literatura que endereçam o problema da fragmentação de dados em XML generaliza os operadores de fragmentação relacionais para o modelo de dados XML (BREMER; GERTZ, 2003; MA et al., 2003; MA; SCHEWE, 2003). Em (MA et al., 2003; MA; SCHEWE, 2003), foram apresentados três tipos de fragmentação: vertical, horizontal e split. Os operadores de fragmentação vertical e horizontal são generalizações dos operadores relacionais, enquanto que o operador split é baseado em um trabalho anterior do mesmo grupo, feito sobre o modelo orientado a objetos (SCHEWE, 2002).

A operação split, no contexto de bases de dados orientadas a objetos, resulta na substituição de uma classe por duas classes, uma referenciando a outra (SCHEWE, 2002). Uma expressão complexa dentro da classe original é substituída por uma referência a uma nova classe. No modelo de dados XML, a semântica dessa operação consiste em substituir um elemento em um dado documento XML por uma referência a um novo elemento (MA; SCHEWE, 2003). Mais especificamente, é criado um novo elemento na raiz do documento XML sendo fragmentado, e um dos antigos elementos passa a referenciar este novo elemento, utilizando para isso atributos do tipo ID, REF ou REFS.

A operação de fragmentação horizontal, conforme proposta em (MA; SCHEWE, 2003), consiste em substituir um determinado elemento em um documento XML por um conjunto de elementos, tendo cada um dos elementos desse conjunto a mesma definição do elemento original. Cada novo elemento criado é associado a um predicado de seleção. O conjunto de predicados de seleção para uma determinada operação deve ser disjunto, de modo que cada instância do elemento original seja movida para apenas um dos novos elementos. É possível também dividir fisicamente o documento XML original em vários documentos, de acordo com a operação de fragmentação realizada.

A operação de fragmentação vertical consiste em substituir um determinado elemento no documento XML original por um conjunto de novos elementos (MA; SCHEWE, 2003). Cada um dos novos elementos contém um subconjunto dos elementos-filho do elemento original. O mesmo ocorre com seus atributos, os quais são distribuídos entre os novos elementos. Essa operação é realizada através de consultas de projeção. Para que essa operação seja reversível, deve ser incluído um atributo do tipo ID em cada um dos novos elementos. Assim, o documento XML original pode ser reconstruído através da operação de junção apropriada. Da mesma forma como na fragmentação horizontal, é possível dividir fisicamente o documento

XML original em vários documentos.

Em (BREMER; GERTZ, 2003), é proposto um método de fragmentação e é esboçado um modelo de alocação de fragmentos XML distribuídos. Neste trabalho, a especificação dos fragmentos XML é feita através de uma sublinguagem do XPath, chamada de *XF*. Uma especificação de fragmento consiste em dois componentes: um fragmento de seleção e um conjunto opcional de fragmentos de exclusão. Ambos os tipos de fragmentos são definidos através de expressões *XF*. Como o *XF* não oferece suporte para predicados de seleção (branching), apenas a fragmentação vertical é considerada. Uma possível extensão do *XF* de modo a suportar predicados de seleção levaria a operação de fragmentação horizontal.

Já em (BOSE et al., 2003), é proposta uma abordagem para tratar o problema de processar consultas sobre um stream de dados XML. Nesta abordagem, o stream de dados é fragmentado em pedaços gerenciáveis de informação. Esses pedaços, ou fragmentos, são relacionados uns com os outros, e podem ser reorganizados no lado cliente após a chegada. O objetivo desta proposta é processar consultas ad-hoc contra esses fragmentos em tempo real e produzir os resultados. Os fragmentos XML seguem o modelo *hole-filler*, no qual todo fragmento é tratado como um *filler* e é associado a um identificador único (ID). Quando o fragmento precisa se referir a outro fragmento em um relacionamento do tipo pai-filho, ele inclui um *hole*, cujo ID casa com o ID do fragmento *filler* correspondente. No lado servidor, um documento XML pode ser fragmentado podando recursivamente a árvore de dados, inserindo um *hole* em cada ponto em que a árvore foi podada e associando ao mesmo um ID. Outra informação importante enviada pelo servidor é a estrutura de tags (*tag structure*) do documento sendo transmitido. Isso é feito no próprio *stream* de dados, como um fragmento separado. Essa estrutura de tags provê a estrutura do documento XML sendo transmitido, e captura todos os caminhos válidos nos dados.

Entre os trabalhos estudados, todos eles definem seus operadores sobre o modelo de dados hierárquico XML (BOSE et al., 2003; BREMER; GERTZ, 2003; MA et al., 2003; MA; SCHEWE, 2003). Esta decisão leva a problemas que são específicos deste modelo de dados. Diferentes fontes XML pertencentes a um mesmo domínio de aplicação podem implementar relacionamentos do tipo muitos-para-muitos de diferentes maneiras. Uma solução mais adequada para o problema da fragmentação de fontes XML seria definir essas operações sobre um modelo de dados do tipo grafo que represente a integração das várias fontes. Porém, nenhum dos trabalhos publicados na literatura sobre fragmentação XML segue esta abordagem.

2.2 Decomposição de Consultas

A decomposição de consultas é definida como a quebra de uma consulta global em subconsultas locais que podem ser executadas nas bases individuais (OZSU; VALDURIEZ, 1991). Posteriormente, os resultados obtidos a partir das subconsultas devem ser combinados de modo que o resultado possa ser construído.

Existem diversos trabalhos na literatura que tratam o problema de decomposição de consultas. O objetivo desta seção é apresentar as diversas propostas, focando nas diferenças de abordagem com relação àquela apresentada neste trabalho. Esta seção está dividida em duas partes: primeiro, será discutido o processo de decomposição de consultas em bases relacionais distribuídas. Após, serão apresentados trabalhos

mais recentes que tratam do problema da decomposição de consultas em fontes XML ou outros tipos de bases de dados.

2.2.1 Decomposição de Consultas em Bases Relacionais Distribuídas

No esquema proposto por Özsu (OZSU; VALDURIEZ, 1991), a decomposição de um consulta distribuída é dividida em duas etapas: a tradução da consulta e a localização dos dados. A tradução da consulta consiste no mapeamento de uma consulta global expressa em cálculo distribuído em uma consulta global expressa em álgebra relacional, utilizando para isso das mesmas técnicas de um SGBD centralizado. A etapa de localização dos dados toma como entrada a consulta algébrica global e aplica informações sobre distribuição de dados à mesma, a fim de localizar seus dados. A localização de dados determina quais fragmentos estão envolvidos na consulta, e assim transforma uma consulta global em uma consulta de local. A etapa que diz respeito mais diretamente ao trabalho aqui apresentado é a de localização de dados, portanto, será dado um enfoque na descrição da mesma ao longo desta Seção.

A camada de localização de dados é responsável por converter uma consulta algébrica sobre relações globais em uma consulta algébrica sobre fragmentos físicos. A localização utiliza informações armazenadas em um esquema de fragmentos. O objetivo da camada de localização é reconstruir uma relação global através da aplicação de regras de reconstrução e pela derivação de um programa de álgebra relacional cujos operandos são os fragmentos. Isso se denomina programa de localização.

Uma consulta localizada é uma consulta em que cada relação global é substituída pelo seu programa de localização. O programa de localização para uma relação fragmentada horizontalmente consiste na união dos fragmentos. Por exemplo, se a relação *Cliente* está dividida em três fragmentos, $Cliente_1$, $Cliente_2$ e $Cliente_3$, o programa de localização para esta relação seria o seguinte:

$$Cliente_1 \cup Cliente_2 \cup Cliente_3$$

A fragmentação vertical distribui uma relação com base em atributos de projeção. Portanto, o programa de localização para fragmentação vertical consiste na junção dos fragmentos. Por exemplo, se a relação *Usuario* está dividida em dois fragmentos, $Usuario_1$ e $Usuario_2$, o programa de localização para esta relação seria o seguinte:

$$Usuario_1 \bowtie Usuario_2$$

A consulta obtida deste modo é chamada de consulta localizada. Sobre esta consulta ainda podem ser feitas reestruturações e simplificações, como por exemplo remover fragmentos que contradizem um determinado operador de seleção e que produzirá como resultado uma relação vazia. Porém, este processo está fora do escopo deste trabalho e não será discutido aqui.

2.2.2 Decomposição de Consultas em Fontes XML e Outros

Os trabalhos propostos na literatura para tratar o problema da decomposição de consultas em múltiplas fontes de dados adotam diferentes arquiteturas, linguagens de consulta e integram diferentes tipos de fontes de dados. Entre as arquiteturas

propostas está a de sistemas mediadores (WIEDERHOLD, 1992). Sistemas mediadores oferecem uma camada de software entre os usuários da aplicação e as fontes de dados, centralizando a tradução da consultas e integração das fontes.

A maior parte dos trabalhos estudados adota a abordagem de mediadores (JOSIFOVSKI; RISCH, 2002; MACKINNON; MARWICK; WILLIAMS, 1998; MANOLESCU; FLORESCU; KOSSMANN, 2001; SATTLER; GEIST; SCHALLEHN, 2005). Nestes trabalhos, as consultas são submetidas contra um modelo global, e posteriormente traduzidas ou decompostas para as fontes individuais. No sistema de integração Agora (MANOLESCU; FLORESCU; KOSSMANN, 2001), o modelo global é XML. Utilizar um modelo global XML para integração de fontes XML pode simplificar o processo de tradução de consultas, porém apresenta um problema maior: como representar em um modelo hierárquico construções semânticas do tipo muitos-para-muitos. Este problema pode ser resolvido através da utilização de um modelo global com maior poder de expressão. Isto é o que foi feito no mediador Yacob (SATTLER; GEIST; SCHALLEHN, 2005), no qual foi definido um modelo global do tipo grafo baseado em RDF. Este modelo define conceitos e relacionamentos entre conceitos, e tem como objetivo prover uma camada de abstração de mais alto nível do que o das fontes de dados. No AMOSII (JOSIFOVSKI; RISCH, 2002), é apresentado um modelo orientado a objetos baseado no DATAPLEX (SHIPMAN, 1981), enquanto que em (MACKINNON; MARWICK; WILLIAMS, 1998) o modelo global é um KBS (Knowledge-based System), contendo uma ontologia de conceitos.

Com relação às linguagens de consulta utilizadas, os trabalhos estudados variam bastante. No sistema de integração Agora (MANOLESCU; FLORESCU; KOSSMANN, 2001), as consultas são escritas em XQuery, o que é uma escolha natural, visto que o modelo global é XML. Já no mediador Yacob (SATTLER; GEIST; SCHALLEHN, 2005), é proposta uma nova linguagem de consulta, CQuery, baseada em XQuery. Entre as principais diferenças com relação a XQuery, a CQuery permite que sejam referenciados conceitos do modelo global. Porém, também é possível referenciar explicitamente elementos XML das fontes locais. Ou seja, é possível misturar elementos do modelo global com elementos das fontes individuais em uma única consulta CQuery. No AMOSII (JOSIFOVSKI; RISCH, 2002), também é definida uma nova linguagem de consulta, chamada AMOSQL. Essa linguagem é similar ao OQL e baseada no OSQL (LYNGBAEK, 1991), e permite que se expresse consultas sobre o modelo global orientado a objetos. Em (MACKINNON; MARWICK; WILLIAMS, 1998), as consultas são escritas através de uma interface restrita (*constrained user interface*), a qual oferece restrições com relação ao tipo, natureza e conteúdo das consultas.

Nem todos os trabalhos estudados integram exclusivamente fontes XML. O Agora e o AMOSII integram, além de XML, outros tipos de fontes de dados, como por exemplo bases relacionais. O trabalho proposto em (MACKINNON; MARWICK; WILLIAMS, 1998) integra apenas bases relacionais. O Yacob integra fontes XML, sendo, de todos os trabalhos estudados, aquele cuja abordagem mais se aproxima daquela desenvolvida neste trabalho. Porém, existem algumas diferenças de enfoque, as quais serão detalhadas a seguir.

O Agora adota uma abordagem *top-down*, também conhecida como LAV (*local-as-view*) para o processo de integração. Isto significa que as fontes individuais são definidas como visões do esquema global. O processo de decomposição envolve uma fase de normalização da consulta global XQuery e, posteriormente, a tradução

dessa consulta para SQL contra o modelo global genérico, o qual é representado através de tabelas relacionais. Finalmente, a consulta SQL é reescrita para as fontes individuais, utilizando para isso as visões através das quais as fontes foram definidas. Esta consulta é então submetida contra *wrappers* responsáveis por acessar cada fonte.

O mediador Yacob (SATTLER; GEIST; SCHALLEHN, 2005) apresenta uma abordagem que é similar, em muitos aspectos, àquela apresentada neste trabalho. A principal diferença é que o Yacob é um mecanismo (*engine*) completo de execução de consultas, contendo módulos em sua arquitetura para funcionalidades como cache e indexação, enquanto que a abordagem apresentada neste trabalho é um algoritmo para decomposição de consultas. As consultas CQuery são efetivamente executadas no mecanismo, o qual é responsável pela implementação dos operadores desta linguagem. Este trabalho, por outro lado, produz uma consulta local XQuery, mas não se responsabiliza pela execução da mesma, tarefa esta que pode ser feita por uma das diversas implementações XQuery existentes. O enfoque no mediador Yacob é muito mais no sentido de produzir um plano de execução barato em termos de esforço de processamento. Pouca atenção é dada ao processo de decomposição propriamente dito, não estando claro, até onde vai nosso conhecimento, qual a classe de problemas tratável pela abordagem apresentada.

Uma outra arquitetura proposta para resolver o problema de consultar fontes XML distribuídas é a *peer-to-peer*. Em (ABITEBOUL et al., 2003), é proposta uma abordagem para execução de consultas em documentos XML dinâmicos, i.e. documentos XML que contém alguns elementos, chamados elementos função, os quais tem um significado especial e representam chamadas a funções em *webservices*. Nesta abordagem, não existe informação de esquema global. Cada fonte é responsável por executar apenas parte da consulta, delegando às demais, i.e. as fontes para as quais estão apontadas as chamadas de função, o restante da mesma. Este comportamento é recursivo, não existindo qualquer forma de controle centralizado, como é o caso dos sistemas mediadores.

Dentre os trabalhos estudados, nenhum deles se baseia em um estudo sobre fragmentação das fontes de dados ao desenvolver o mecanismo de decomposição, apesar de existir na literatura uma série de trabalhos sobre fragmentação em fontes XML (BOSE et al., 2003; BREMER; GERTZ, 2003; MA et al., 2003; MA; SCHEWE, 2003). No contexto de bancos de dados distribuídos, o estudo sobre fragmentação de dados está relacionado ao projeto top-down de uma base distribuída cujos dados estejam fragmentados e estes fragmentos alocados nos sites de tal forma que o processamento de consultas como um todo tenha uma performance razoável. Por outro lado, levar em conta o processo de fragmentação ao definirmos os algoritmos de decomposição torna bastante claro quais os possíveis cenários de distribuição que o algoritmo é capaz de tratar. Os trabalhos estudados, além de não levarem em conta um estudo sobre fragmentação, não propuseram outra alternativa de modo a tornar claro qual a classe de problemas tratada pela abordagem proposta.

2.3 BInXS e CXPath

Nesta Seção, é apresentada a abordagem para integração de esquemas XML BInXS, a qual é utilizada para gerar um modelo global a partir de um conjunto de esquemas XML relativos a um domínio comum. A seguir, é apresentada a linguagem

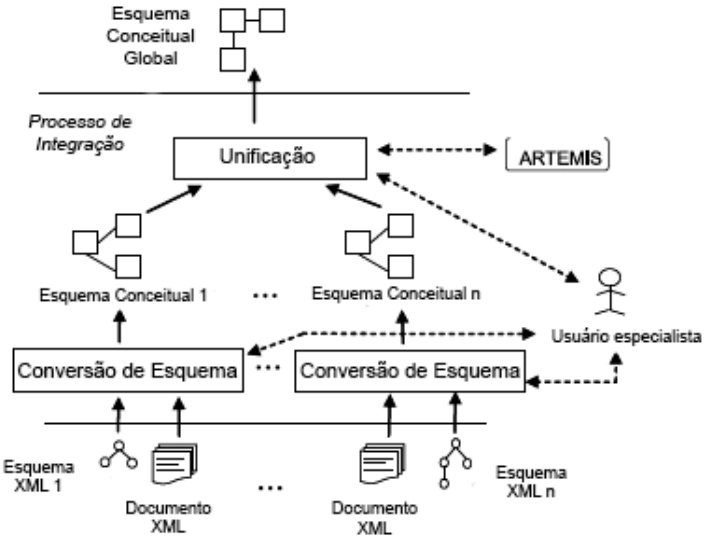

Figura 2.1: Arquitetura do BInXS (MELLO; HEUSER, 2005)

de consulta CXPath, utilizada para expressar consultas contra o modelo global, juntamente com um mecanismo para tradução de uma consulta global CXPath em uma consulta local XPath.

2.3.1 BInXS

O BInXS é uma processo *bottom-up* e semi-automático para integração de esquemas XML, desenvolvido por nosso grupo de pesquisas (MELLO; HEUSER, 2005). O processo proposto é semi-automático pois requer a intervenção de um especialista humano para validar a integração semântica dos dados, e é *bottom-up* pois ele gera um esquema global a partir de um conjunto de esquemas XML, sendo classificada como uma abordagem de integração *global-as-view* (GAV). O esquema global gerado abstrai a elevada heterogeneidade das fontes de dados XML e leva em consideração a intenção semântica de todas as fontes.

Esta abordagem tem duas fases, conforme pode ser visto na figura 2.1: *conversão de esquemas* e *unificação*. A fase de conversão de esquemas é responsável por mapear cada esquema lógico XML em um esquema conceitual correspondente. O esquema XML é mapeado para um esquema canônico pois o mesmo provê um alto nível de abstração para os dados XML. Além disso, um mesmo esquema conceitual pode abstrair diversos esquemas lógicos XML para um mesmo domínio de aplicação. A fase de unificação pega um conjunto de esquemas conceituais, gerados pela fase anterior, e executa uma integração semântica, criando um esquema conceitual global. A intervenção de um usuário especialista é requerida em ambas as fases. Para maiores detalhes relativo ao processo de integração, consulte (MELLO; HEUSER, 2005).

Informações de mapeamento são definidas durante a etapa de conversão para cada conceito e relacionamento no esquema conceitual. O BInXS adota expressões XPath para especificar mapeamentos para um esquema XML. O mapeamento para um conceito é definido através de uma expressão de caminho absoluta do XPath. Já o mapeamento de um relacionamento é definido através de uma expressão de

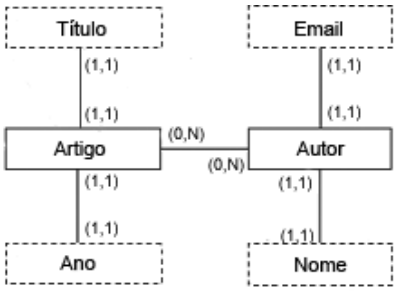

Figura 2.2: Exemplo de modelo conceitual

caminho relativa. Mapeamentos relativos são definidos em ambas as direções do relacionamento, de modo a possibilitar a tradução de qualquer travessia sobre o esquema conceitual.

2.3.2 CXPath

CXPath (CAMILLO; MELLO; HEUSER, 2003) é uma linguagem de consulta, baseado na sintaxe do XPath (W3C, 1998), a qual é utilizada para construir consultas sobre o esquema conceitual global. Uma consulta CXPath especifica uma expressão de caminho do tipo XPath para alcançar a informação que se deseja recuperar, com predicados de seleção opcionais que podem ser definidos sobre este caminho.

Apesar de ser baseado no XPath, o CXPath e o XPath tem semânticas diferentes, pois são aplicados sobre diferentes modelos de dados. No CXPath, são referenciados nomes de conceitos, ao invés de nomes de elementos XML. O uso do operador de navegação (barra) no CXPath serve para navegar para conceitos relacionados, ao invés de conceitos filho, como no XPath. Ao contrário do XPath, o CXPath não possui operadores de navegação para ancestrais (por exemplo, operador "..") ou descendentes (por exemplo, o operador "//").

A tradução de uma expressão CXPath para XPath aplica uma abordagem de reescrita, i.e. cada referência a um conceito, assim como cada relacionamento de travessia encontrado em uma expressão CXPath, são substituídos pela informação de mapeamento correspondente para a fonte XML em questão. O processo de tradução segue basicamente os seguintes passos. A expressão CXPath é analisada da esquerda para a direita. Quando o primeiro conceito de uma expressão CXPath absoluta é encontrado, a expressão XPath que mapeia este conceito para a fonte é escrito na saída. Os demais conceitos encontrados na entrada tem um contexto, i.e. são relativos a algum outro conceito na expressão CXPath sendo traduzida. Neste caso, a expressão XPath que mapeia o relacionamento de travessia do conceito contexto para o conceito relativo é escrito na saída. Para maiores detalhes relativo ao este processo de tradução, consulte (CAMILLO; MELLO; HEUSER, 2003).

Para ilustrar o uso da linguagem CXPath e o processo de tradução de consultas para XPath, serão apresentados alguns exemplos executados contra o modelo conceitual da Figura 2.2, considerando três instâncias XML com os esquemas apresentados na Figura 2.3.

Exemplo 2.1 *Retornar o título dos artigos publicados no ano de 2006.*

```
/Artigo[Ano="2006"]/Título
```

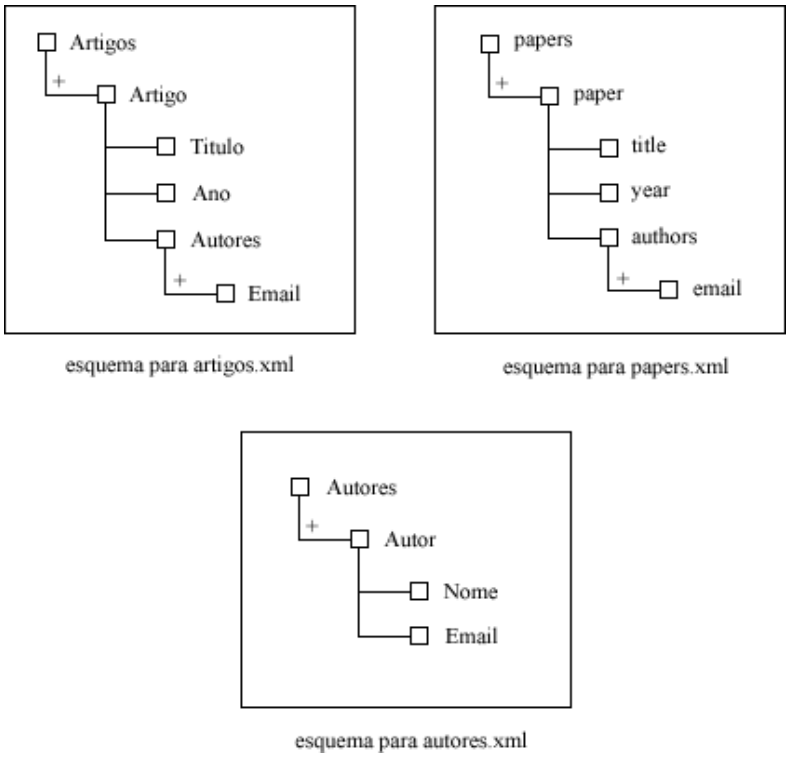

Figura 2.3: Esquemas das instâncias XML

Aplicando o processo de tradução de consultas descrito anteriormente, considerando a fonte `artigos.xml`, o resultado seria o seguinte:

```
/Artigos/Artigo[Ano="2006"]/Titulo
```

Caso a mesma consulta seja traduzida considerando a fonte `paper.xml`, o resultado seria o seguinte:

```
/papers/paper[year="2006"]/title
```

Porém, não é possível considerar ambas as fontes simultaneamente no processo de tradução apresentado acima. Para considerar ambas as fontes simultaneamente, seria necessário produzir uma união de ambas as expressões. Esse tipo de operação não é tratado pela abordagem apresentada em (CAMILLO; MELLO; HEUSER, 2003).

Exemplo 2.2 *Retornar o nome dos autores que publicaram artigos no ano de 2004.*

```
/Artigo[Ano="2004"]/Autor/Nome
```

Não é possível efetuar a tradução desta consulta através da abordagem descrita acima, pois a mesma envolve as três fontes, incluindo uma junção entre essas fontes. Não seria possível sequer traduzir a consulta considerando apenas uma única fonte, como foi feito no exemplo anterior, pois nenhuma das fontes possui mapeamentos para todos os conceitos envolvidos nesta consulta.

Tratar as limitações dessa abordagem, expostas através dos exemplos acima, é um dos objetivos deste trabalho.

Tabela 2.1: Comparação entre as abordagens para consultas distribuídas

	Agora	AMOSII	Yacob	*Abiteboul*	*Mackinnon*
Arquitetura utiliza pela abordagem	Mediada	Mediada	Mediada	Peer-to-peer	Mediada
Modelo de dados do esquema global	XML	extensão OO do DATAPLEX	variante do RDF	n.a.	KBS
Linguagem de consulta	XQuery	AMOSQL	CQuery	$XQuery_{dr}$	Interface restrita para o usuário
Modelo de dados das fontes integradas	XML e relacional	XML, relacional e outros AMOSII	XML	XML	Relacional

2.4 Conclusão

Existe uma série de trabalhos na literatura que endereça os problemas da fragmentação e decomposição de consultas em múltiplas fontes XML. A tabela 2.1 apresenta um resumo das principais características das abordagens estudadas para decomposição de consultas.

Os trabalhos que tratam o problema da fragmentação em fontes XML definem seus operadores sobre o modelo XML, ao invés de considerarem um modelo global do tipo grafo que integre os esquemas das diversas fontes. Casos como representações de uma mesma informação conceitual em diferentes documentos XML, com diferentes estruturas, não são tratados. Cada documento, mesmo que contendo a mesma informação conceitual (como por exemplo, os documentos `artigos.xml` e `papers.xml`, que contém os mesmos dados, porém possuem estruturas diferentes) são considerados como diferentes documentos. Relacionamentos conceituais do tipo muitos-para-muitos também não são tratados por essas abordagens.

Com relação aos trabalhos sobre decomposição de consultas em fontes XML, nenhum deles se baseia em um estudo sobre o projeto de fragmentação em XML, seguindo a abordagem tradicional e bem fundamentada da decomposição de consultas em bases relacionais. Os principais ganhos ao seguir esta abordagem são: (i) possuir um vasto fundamento teórico em bases relacionais, que pode ser generalizado para o modelo de dados XML, (ii) possuir um arcabouço formal para descrever claramente quais são os casos tratáveis pela abordagem apresentada.

O BInXS e a linguagem de consulta CXPath, juntamente com o mecanismo de tradução apresentado em (CAMILLO; MELLO; HEUSER, 2003), oferecem um bom ponto de partida para desenvolver uma abordagem que integre o estudo sobre fragmentação e um algoritmo para decomposição de consultas em fontes XML. Para atingir esse objetivo, é necessário resolver dois problemas:

- Redefinir os operadores de fragmentação apresentados na literatura para o modelo conceitual e base conceitual gerados através da abordagem BInXS.
- Estender o mecanismo de tradução do CXPath, de modo a possibilitar a tradução de consultas considerando múltiplas fontes XML, ao invés de apenas uma.

Nos próximos capítulos, é apresentada uma abordagem que se propõe a resolver estes problemas.

3 MODELO DE DADOS

Este Capítulo contém uma definição formal do modelo conceitual e da correspondente base conceitual que são utilizadas na abordagem proposta neste trabalho.

3.1 Modelo Conceitual

O modelo conceitual é uma abstração de alto nível dos esquemas das fontes de dados XML. Ele é construído através de uma abordagem *bottom-up*. Inicialmente, cada esquema local XML é abstraído para um esquema conceitual local, e após os esquemas conceituais locais são integrados, resultando em um esquema conceitual global (MELLO; CASTANO; HEUSER, 2002; MELLO; HEUSER, 2001).

O modelo conceitual é uma versão simplificada do modelo ORM (HALPHIN, 1998). Ele define *conceitos* e *relacionamentos* entre os conceitos. Existem dois tipos de conceitos: *léxicos* e *não-léxicos*. Conceitos léxicos representam objetos que possuem conteúdo textual. Elementos léxicos abstraem elementos XML atômicos, como elementos #PCDATA ou valores de atributos. Conceitos não-léxicos não possuem uma representação textual direta, e são abstrações de elementos XML que contém outros elementos. Um conceito não-léxico pode possuir um *identificador*. Um identificador é um conjunto de relacionamentos que unicamente identifica uma instância do conceito. Relacionamentos no nível conceitual são abstrações de dois tipos de construção no nível XML. Um relacionamento pode possuir uma implementação *navegacional*, i.e. pode representar um relacionamento de caminho de acesso no nível XML. Um relacionamento também pode possuir uma implementação *associativa*, i.e. pode representar um identificador de referência entre dados de duas fontes XML diferentes. Para detalhes sobre o processo de abstração de um esquema XML em um esquema conceitual, por favor consulte (MELLO; HEUSER, 2001).

A Figura 3.1 mostra um exemplo de esquema conceitual para matrícula de estudantes em uma universidade. Existem três conceitos não-léxicos: *Curso*, *Matrícula* e *Estudante*, os quais são representados por retângulos sólidos. Cada um desses conceitos está relacionado a pelo menos um conceito léxico (retângulo tracejado). Por exemplo, *Estudante* é relacionado a *Nome* e *Número*. O diagrama representa também a cardinalidade dos relacionamentos entre os conceitos. Por exemplo, neste modelo, *Curso* e *Matrícula* possuem um relacionamento um-para-muitos. Os relacionamentos identificadores estão rotulados com 'ID' (por exemplo, *Estudante* é identificado pelo relacionamento com *Número*).

O esquema na Figura 3.1 pode possuir várias implementações diferentes em XML. Na Figura 3.1 duas instâncias XML com diferentes esquemas mas correspondentes ao mesmo esquema conceitual estão representadas: uma que implementa o relaciona-

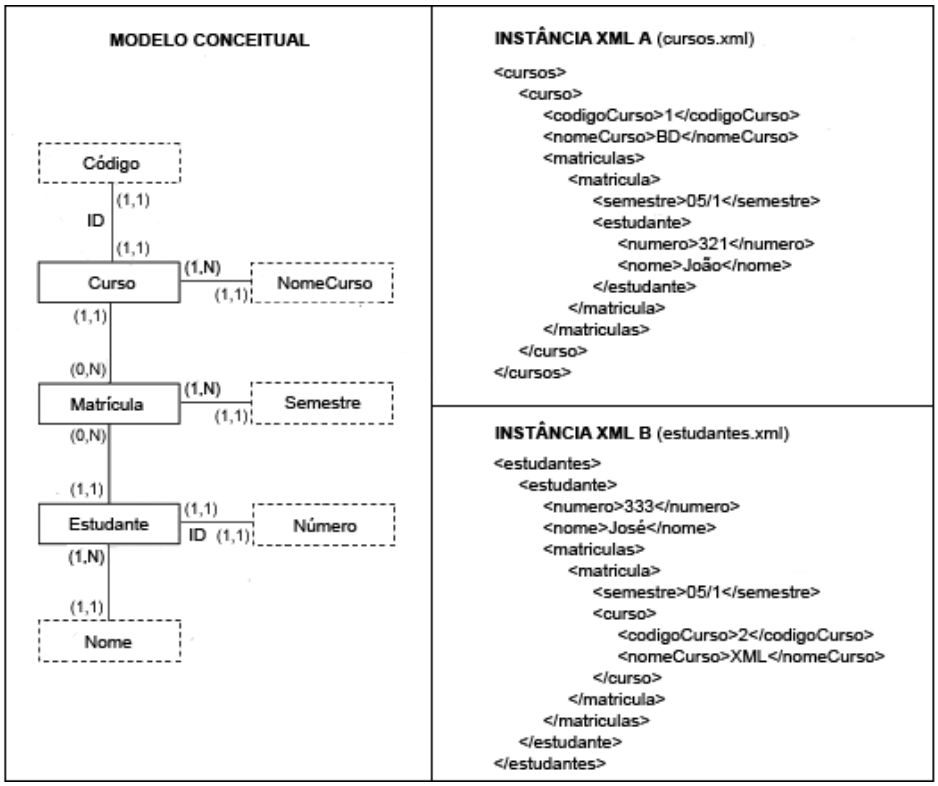

Figura 3.1: Um esquema conceitual e algumas instâncias XML

mento muitos-para-muitos entre estudante e curso de uma perspectiva do estudante, i.e., o elemento estudante contém os cursos nos quais ele está matriculado, e a outra que implementa o relacionamento de uma perspectiva do curso, i.e., o elemento curso contém os estudantes que estão matriculados no mesmo.

Uma definição formal do modelo conceitual é dada na Definição 3.1.

Definição 3.1 *Um modelo conceitual CM (Conceptual Model) é uma tupla* $CM =< NL, L, R >$*, onde:*

- *NL (non-lexical) é um conjunto de conceitos não-léxicos.*
- *L (lexical) é um conjunto de conceitos léxicos.*
- *R (relationship) é um conjunto de relacionamentos entre os conceitos. Um relacionamento* $r \in R$ *é uma tupla* $< c_1, c_2, c_d, c_i, [n] >$*, onde* c_1 *e* c_2 *são conceitos, sendo* $c_1 \in NL$ *e* $c_2 \in NL \cup L$*, e* c_d *e* c_i *são as cardinalidades, onde* c_d *é a cardinalidade direta - de* c_1 *para* c_2 *- e* c_i *é a cardinalidade inversa - de* c_2 *para* c_1*. Uma cardinalidade* c_d *ou* c_i *é uma tupla* $< min, max >$*, onde min é a cardinalidade mínima e max é a cardinalidade máxima. n é o nome opcional do relacionamento.*
- *A função* $ID(c) \rightarrow ID_c$ *retorna um conjunto de identificadores de um dado conceito* $c \in NL$*. Um identificador* $id \in ID_c$ *é um relacionamento* $r \in R$*, onde* $r.c_1 = c$*,* $r.c_2 \in L$ *e* $r.c_i = 1$*.*

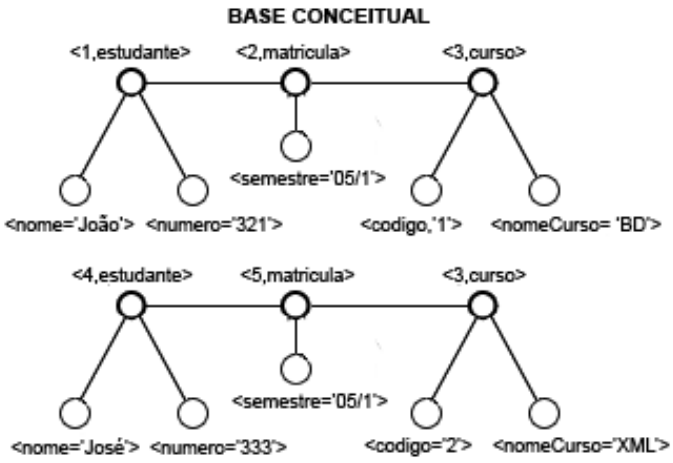

Figura 3.2: Um exemplo de base conceitual

3.2 Base Conceitual

A base conceitual é uma base de dados abstrata, a qual contém instâncias de um dado esquema conceitual. É uma representação abstrata de um documento XML.

Os conceitos não-léxicos, léxicos e relacionamentos de um dado esquema conceitual podem possuir diversas instâncias em uma determinada base conceitual. A Figura 3.2 mostra um exemplo de base conceitual para o esquema conceitual apresentado na Figura 3.1. Os pequenos círculos representam instâncias de conceitos, enquanto as linhas representam instâncias de relacionamentos.

Uma definição formal da base conceitual é dada na Definição 3.2.

Definição 3.2 *Uma base conceitual de um dado modelo conceitual CM é uma tupla* $CM_B =< NL_B, L_B, R_B >$, *onde:*

- NL_B *é o conjunto de instâncias não-léxicas. Uma instância não-léxica* $nl \in NL_B$ *é uma tupla* $< oid, c >$, *onde oid é o identificador do objeto e* $c \in CM.NL$.

- L_B *é o conjunto de instâncias léxicas. Uma instância léxica* $l \in L_B$ *é uma tupla* $< c, v >$, *onde* $c \in CM.L$ *e v é o valor.*

- R_B *é o conjunto de instâncias de relacionamentos. Uma instância de relacionamento* $r \in R_B$ *é uma tupla* $< i_1, i_2 >$, *onde* $i_1 \in NL_B$, $i_2 \in NL_B \cup L_B$, $e < i_1.c, i_2.c > \in CM.R$.

- *A base conceitual* CM_B *obedece às restrições de identificadores e cardinalidade definidas no modelo conceitual CM.*

4 OPERADORES DE FRAGMENTAÇÃO

Diversos autores na literatura endereçam o problema de fragmentação de documentos XML (ABITEBOUL et al., 2003; BOSE et al., 2003; BREMER; GERTZ, 2003; MA et al., 2003; MA; SCHEWE, 2003). Todavia, em nenhum dos trabalhos propostos é considerado um modelo global do tipo grafo na definição dos operadores de fragmentação.

Uma abordagem natural para tratar o problema de fragmentação de documentos XML é olhar para o trabalho existente em projeto de distribuição de bases de dados relacionais (OZSU; VALDURIEZ, 1991) e orientadas a objetos e adaptar os conceitos propostos para o modelo de dados XML. Isto é o que foi feito em (MA et al., 2003; MA; SCHEWE, 2003), onde os operadores de fragmentação vertical e horizontal de bases de dados relacionais (CERI; NEGRI; PELAGATTI, 1982; NAVATHE et al., 1984) foram adaptados para o modelo de dados XML. Adicionalmente, um novo operador de fragmentação, chamado fragmentação split, foi generalizado do modelo de dados orientado a objetos (SCHEWE, 2002).

Neste Capítulo, uma especificação de distribuição para documentos XML que são descritos por um modelo conceitual do tipo grafo é apresentada. Em contraste com a abordagem de Schewe (MA et al., 2003; MA; SCHEWE, 2003), os operadores de fragmentação são aplicados sobre o modelo conceitual e a base conceitual, ao invés de documentos XML. O resultado do processo de fragmentação é um conjunto de esquemas conceituais locais e bases conceituais locais, as quais são diretamente mapeadas para documentos XML através do processo descrito em (MELLO; HEUSER, 2001).

A abordagem proposta neste trabalho tem a vantagem de claramente separar dois problemas distintos, o problema da fragmentação e o problema de abstrair os detalhes de implementação em XML. Se os operadores de fragmentação forem aplicados diretamente no nível do esquema XML, será necessário tratar o problema de que uma mesma informação conceitual pode possuir diferentes representações no nível XML. Nesta abordagem, documentos que possuem diferentes representações mas possuem conceitualmente o mesmo conteúdo serão descritas por um único esquema conceitual local.

Este Capítulo adota a seguinte estrutura. Inicialmente, é apresentada uma definição informal e intuitiva dos conceitos, juntamente com um exemplo. Em seguida, é apresentada uma definição formal dos mesmos. Nas Seções abaixo, são definidos inicialmente os esquemas locais que são obtidos a partir dos operadores de fragmentação. Após, são discutidos os operadores de fragmentação. Por fim, são apresentadas as conclusões.

4.1 Modelo Conceitual Local

Um modelo conceitual local é um subconjunto de um modelo conceitual global. Um esquema conceitual global pode ser fragmentado em vários esquemas conceituais locais através de operadores de fragmentação. Os operadores que irão produzir esquemas conceituais locais são os de fragmentação vertical e fragmentação split, os quais serão detalhados abaixo. Após a fragmentação, deve ser possível reconstruir o esquema conceitual a partir de diversos esquemas locais. Isto significa que os esquemas locais devem conter algumas informações específicas de forma a permitir essa reconstrução.

Quando dois conceitos não-léxicos que estão associados através de um relacionamento no esquema conceitual global são fragmentados em dois esquemas locais diferentes, *identificadores de referência* devem ser incluídos nesses esquemas locais. Um identificador de referência é uma informação que implementa um relacionamento entre dois esquemas conceituais locais da mesma forma que uma chave estrangeira implementa um relacionamento entre duas relações em uma base de dados relacional. O esquema local em que o identificador de referência é incluído depende da cardinalidade do relacionamento. Cada identificador de referência é mapeado para um identificador do conceito não-léxico relacionado. Isto será explicado em detalhes na fragmentação split.

A Figura 4.1 mostra um exemplo de esquema conceitual global e a Figura 4.2 mostra esquemas locais que podem ser obtidos a partir da fragmentação deste esquema conceitual. Os conceitos não-léxicos *Matrícula*, *Curso*, *Professor* e *Leciona*, os quais são relacionados no esquema conceitual exibido na figura 4.1, foram fragmentados nos esquemas conceituais locais da figura 4.2. Alguns relacionamentos no esquema global foram implementados nos esquemas locais através de identificadores de referência. Por exemplo, pegue o relacionamento entre *Matrícula* e *Curso* que aparece no esquema global. Como os conceitos *Matrícula* e *Curso* estão distribuídos em esquemas locais diferentes (LCM_3 e LCM_2 respectivamente), o relacionamento no nível conceitual global deve ser implementado através de um identificador de referência no nível local. Neste exemplo, o conceito *CódigoCurso* foi adicionado no esquema local LCM_3. As instâncias deste conceito são referências às instâncias do identificador do conceito *Curso* no esquema local LCM_2. A mesma idéia é válida para o conceito *EmailProfessor* no LCM_2.

Para a correta implementação de relacionamentos entre diferentes esquemas locais através de identificadores de referência, é necessário inicialmente converter no esquema global os relacionamentos do tipo muitos-para-muitos em relacionamentos do tipo um-para-muitos. Esta conversão deve ser realizada adicionando-se uma nova entidade no esquema, a qual implementará o relacionamento muitos-para-muitos através de dois relacionamentos um-para-muitos, da mesma forma como é feito na conversão de um modelo ER para um modelo relacional. No exemplo da Figura 4.1, o relacionamento muitos-para-muitos entre *Curso* e *Estudante* é implementado através do conceito *Matrícula*. A abordagem proposta neste trabalho supõe que este passo tenha sido previamente realizado, não tratando casos de relacionamentos muitos-para-muitos diretamente.

Uma definição formal do modelo conceitual local é dada nas Definições 4.1 e 4.2.

Definição 4.1 *Um sub modelo conceitual CM' de um dado modelo conceitual global*

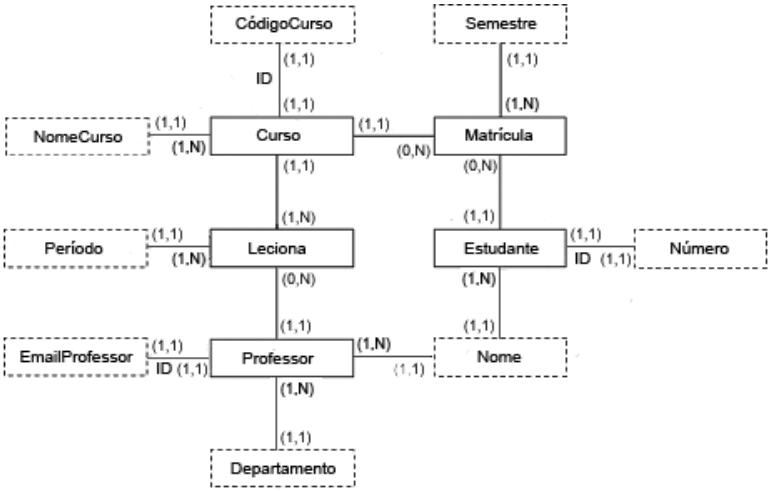

Figura 4.1: Esquema conceitual global para os exemplos de fragmentação

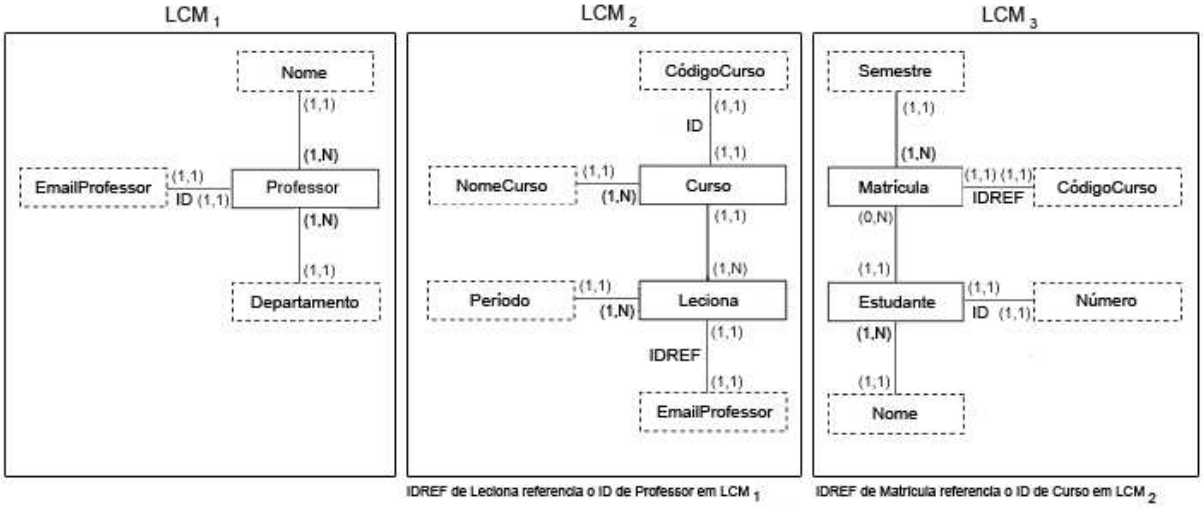

Figura 4.2: Esquemas conceituais locais produzidos através do operador de fragmentação split

GCM (Global Conceptual Model) $GCM =< NL, L, R >$ *é uma tupla* $CM' =< NL', L', R' >$*, onde:*

- $CM'.NL' \subseteq GCM.NL$.
- $CM'.L' \subseteq GCM.L$.
- $CM'.R' \subseteq GCM.R$. *Para todos os relacionamentos* $r \in CM'.R'$, $r.c_1 \in CM'.NL'$, $r.c_2 \in CM'.NL' \cup CM'.L'$.
- $CM'.ID(nl') \subseteq GCM.ID(nl)$, *onde* $nl' \in CM'.NL'$, $nl \in GCM.NL$, *e* $LCM'.ID(nl') \subseteq CM'.R'$.

Definição 4.2 *Um modelo conceitual local LCM (Local Conceptual Model) é uma tupla* $LCM =< CM', IDREF >$*, onde:*

- *CM' é um sub modelo conceitual de um dado modelo conceitual global CM.*

- *IDREF (Identifier References) é um conjunto de identificadores de referência. Um identificador de referência* $idref \in LCM.IDREF$ *é uma tupla* $< ref, DEST >$, *onde* $ref \in LCM.R'$ *e* $DEST$ *são um conjunto de tuplas* $< LCM_d, id >$, *onde* LCM_d *é o modelo conceitual local de destino, e* $id \subseteq LCM_d.ID(nl)$, *onde* $nl \in LCM_d.NL$.

4.2 Fragmentação Split

O operador de fragmentação split é baseado no operador apresentado em (MA; SCHEWE, 2003), onde um dado elemento de um documento XML é substituído por uma referência a um novo elemento. Este operador teve origem em um trabalho anterior em bancos de dados orientados a objetos (SCHEWE, 2002), no qual uma operação complexa em uma classe é substituída por uma referência a uma nova classe. Neste trabalho, o operador split tem um comportamento diferente. Ele pega um esquema conceitual e divide o mesmo em dois esquemas conceituais locais distintos. Os relacionamentos do esquema global que são divididos em diferentes esquemas locais são implementados através de identificadores de referência. A criação de identificadores de referência depende da cardinalidade dos relacionamentos.

A Figura 4.2 apresenta um exemplo de resultado do operador split aplicado sobre o esquema conceitual da Figura 4.1. Para produzir os três esquemas conceituais locais apresentados neste exemplo, é necessário aplicar o operador split duas vezes. Primeiro, o esquema conceitual global é dividido nos esquemas conceituais locais LCM_1 e LCM_{tmp}. Após, o esquema local LCM_{tmp} é dividido nos esquemas locais LCM_2 e LCM_3.

A primeira operação split é

$$SPLIT(CM, \{\text{Professor}\}) \rightarrow < LCM_1, LCM_{tmp} >$$

O operador split recebe um esquema conceitual (CM) e um conjunto de conceitos não-léxicos ({Professor}), e cria dois esquemas conceituais locais (LCM_1 e LCM_{tmp}). O primeiro esquema local (LCM_1) contém os conceitos não-léxicos especificados como parâmetro ({Professor}), juntamente com todos os conceitos léxicos relacionados a estes conceitos não-léxicos, incluindo identificadores e identificadores de referência, bem como todos os relacionamentos entre eles. Os relacionamentos entre conceitos não-léxicos separados em diferentes esquemas locais são substituídos por identificadores de referência entre os esquemas locais. Neste exemplo, o relacionamento entre *Leciona* e *Professor* que aparece no nível global é substituído por uma referência de $LCM_{tmp}.Leciona$ para $LCM_1.Professor$.

A segunda operação split é

$$SPLIT(LCM_{tmp}, \{Curso, Leciona\}) \rightarrow < LCM_2, LCM_3 >$$

Esta operação resulta no esquema local LCM_2 contendo os conceitos *Curso* e *Leciona* e no modelo local LCM_3 contendo os conceitos remanescentes. O relacionamento entre *Matrícula* e *Curso* que aparece em LCM_{tmp} é substituído por uma referência de $LCM_3.Matrícula$ para $LCM_2.Curso$.

As Definições 4.3 e 4.4 formalizam essas idéias. A Definição 4.3 é uma definição auxiliar que endereça a criação de identificadores de referência. Ela define uma função que implementa um relacionamento global entre dois conceitos não-léxicos,

c_1 and c_2, em dois esquemas conceituais locais, através da criação de identificadores de referência. O resultado desta função é um esquema conceitual local com um conjunto de identificadores de referência para o conceito de destino, c_2, localizado em um esquema conceitual local diferente.

Definição 4.3 *Seja GCM um modelo conceitual, r um relacionamento em GCM, e LCM_1 e LCM_2 modelos conceituais locais. A função*

$$generateIDRefs(GCM, r, LCM_1, LCM_2) \rightarrow LCM_1$$

gera identificadores de referência em LCM_1 que implementam o relacionamento r. A Definição 4.4 trata o operador de split.

- $ID(r.c_2).c_2 \subseteq LCM_1.L$.
- $\{r_1, r_2, \ldots, r_n\} \subseteq LCM_1.R$, *onde* $r_i.c_1 = r.c_1$, $r_i.c_2 \in GCM.ID(r.c_2).c_2$, $r_i.c_i =< 1, 1 >$ *e* $r_i.c_d = r.c_i$.
- $\{idr_1, idr_2, \ldots, idr_n\} \subseteq LCM_1.IDREF$, *onde* $idr_i.ref \in LCM_1.R$, $idr_i.DEST.LCM_d = \{LCM_2\}$ *e* $idr_i.DEST.id \subseteq LCM_2.ID(r.c_2)$.

Definição 4.4 *A função $SPLIT(CM, C) \rightarrow < LCM_1, LCM_2 >$ define o operador de fragmentação split, onde CM é um modelo conceitual (local), C é um conjunto de conceitos não-léxicos $C \subseteq CM.NL$, e LCM_1 e LCM_2 são modelos conceituais locais produzidos como resultado, tal que:*

- $LCM_1.NL = C$.
- $LCM_1.L$ *tal que, para todo* $r \in CM.R$, *onde* $r.c_1 \in C$ *e* $r.c_2 \in CM.L$, $r.c_2 \in LCM_1.L$.
- $LCM_1.R$ *tal que, para todo* $r \in CM.R$, *onde* $r.c_1 \in C$ *e* $r.c_2 \in C \cup CM.L$, $r \in LCM_1.R$.
- $LCM_1.ID(nl_1)$ *tal que, para todo* $id \in CM.ID(nl_2)$, *onde* $nl_1 \in LCM_1.NL$, $nl_2 \in CM.NL$, $id.c_1 \in C$, *então* $id \in LCM_1.ID(nl)$.
- $LCM_1.IDREF$ *tal que, se CM é um modelo conceitual local, então para todo* $idref \in CM.IDREF$, *onde* $idref.c_1 \in C$, $idref \in LCM_1.IDREF$.
- $LCM_2.NL = (CM.NL - C)$.
- $LCM_2.L$ *tal que, para todo* $r \in CM.R$, *onde* $r.c_1 \in (CM.NL - C)$ *e* $r.c_2 \in CM.L$, $r.c_2 \in LCM_2.L$.
- $LCM_2.R$ *tal que, para todo* $r \in CM.R$, *onde* $r.c_1 \in (CM.NL - C)$ *e* $r.c_2 \in (CM.NL - C) \cup CM.L$, $r \in LCM_2.R$.
- $LCM_2.ID(nl_2)$ *tal que, para todo* $id \in CM.ID(nl_1)$, *onde* $nl_1 \in CM.NL$, $nl_2 \in LCM_2.NL$, $id.c_1 \in (CM.NL - C)$, *então* $id \in LCM_2.ID(nl_2)$.
- $LCM_2.IDREF$ *tal que, se CM é um modelo conceitual local, então para todo* $idref \in CM.IDREF$, *onde* $idref.c_1 \in (CM.NL - C)$, $idref \in LCM_2.IDREF$.

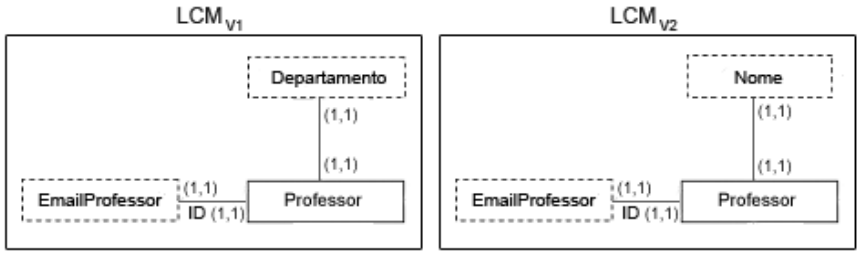

Figura 4.3: Exemplo de fragmentação vertical de LCM_1 (Figura 4.2)

- *para todos relacionamentos* $r \in CM.R$, *onde* $r.c_1 \in LCM_1.NL$ *e* $r.c_2 \in LCM_2.NL$, *então:*
  - *Se* $r.c_d =< 0,1 >$ *e* $r.c_i =< 1,1 >$, $generateIDRefs(CM, r, LCM_2, LCM_1) \subseteq LCM_2$.
  - *Se* $r.c_d =< 0,1 >$ *e* $r.c_i =< 0,1 >$, *então duas definições disjuntas são válidas:*
    1. $generateIDRefs(CM, r, LCM_1, LCM_2) \subseteq LCM_1$.
    2. $generateIDRefs(CM, r, LCM_2, LCM_1) \subseteq LCM_2$.
  - *Se* $r.c_d.max = 1$ *e* $r.c_i.max = N$, *então* $generateIDRefs(CM, r, LCM_1, LCM_2) \subseteq LCM_1$.

4.3 Fragmentação Vertical

A fragmentação vertical é baseada no operador relacional correspondente (NAVATHE et al., 1984), o qual produz fragmentos projetando subconjuntos de atributos. Neste trabalho, cada fragmento contém subconjuntos de conceitos léxicos do modelo conceitual.

A Figura 4.3 mostra um exemplo de fragmentação vertical. O esquema conceitual local LCM_1 na Figura 4.2 é fragmentado em dois esquemas conceituais locais: LCM_{V1} e LCM_{V2}. As operações que produzem estes esquemas são as seguintes:

$$VF(LCM_1, \{Departamento\}) \rightarrow LCM_{V1}$$

$$VF(LCM_1, \{Nome\}) \rightarrow LCM_{V2}$$

Este operador pega um esquema conceitual e um parâmetro C, contendo um conjunto de conceitos léxicos, e produz um esquema conceitual fragmentado LCM_V. O esquema fragmentado contém apenas os conceitos léxicos especificados em C, mais todos os conceitos identificadores e identificadores de referência. A Definição 4.5 formaliza esta idéia.

Observe que não é possível descrever, utilizando os operadores apresentados, o caso de um conceito não-léxico associado ao um mesmo conceito léxico em mais de um esquema local, considerando que o relacionamento entre os mesmos não seja identificador. Isto significa que um conceito não-léxico e um conceito léxico só podem existir associados em mais de um esquema conceitual local caso o relacionamento entre ambos seja identificador. A abordagem apresentada não trata casos que não se enquadrem neste requisito.

DEFINIÇÃO 4.5 *O operador de fragmentação vertical é uma função VF (Vertical Fragmentation)* $VF(CM, C) \rightarrow LCM_V$*, onde* CM *é um modelo conceitual (local),* C *é um conjunto de conceitos léxicos* $C \subseteq CM.L$*, e* LCM_V *é um modelo conceitual local produzido como resultado, tal que:*

- $LCM_V.NL = CM.NL$.
- $r \in LCM_V.R$ *tal que* $r.c_1 \in CM.NL$ *e* $r.c_2 \in (CM.NL \cup C \cup CM.ID(r.c_1).c_2 \cup CM.IDREF.ref.c_2)$.
- $LCM_V.L = (C \cup CM.ID.c_2 \cup CM.IDREF.ref.c_2)$.
- $LCM_V.ID(nl) = CM.ID(nl)$*, para todos conceitos* $nl \in CM.NL$.
- $LCM_V.IDREF = CM.IDREF$*, se* CM *é um modelo conceitual local.*

4.4 Fragmentação Horizontal

Assim como na fragmentação vertical, a fragmentação horizontal é baseada no operador relacional correspondente (CERI; NEGRI; PELAGATTI, 1982), o qual produz fragmentos que correspondem a subconjuntos de tuplas. O operador apresentado neste trabalho produz fragmentos que correspondem a subconjuntos de instâncias de uma dada base conceitual. Cada fragmento é definido através de um predicado de seleção. Se a instância obedece ao predicado, é incluída no fragmento.

Como no operador relacional, este operador produz dois tipos de fragmentos horizontais: primário e derivado. Um fragmento primário contém todas as instâncias selecionadas através de uma aplicação direta do predicado de seleção. Um fragmento derivado inclui todas as instâncias que são relacionadas às instâncias selecionadas no fragmento primário ou a outras instâncias do fragmento derivado.

A Figura 4.4 mostra um exemplo de fragmentação horizontal. Nesta figura, pequenos círculos representam instâncias de conceitos e linhas representam instâncias de relacionamentos. Uma base conceitual exemplo do esquema conceitual local LCM_3 (Figura 4.2) é horizontalmente fragmentada através da seguinte operação:

$$HF(LCM_3, \sigma)$$

onde o predicado de seleção σ é `semestre='2005/1'`. Os nodos pretos na Figura 4.4 representam uma correspondência direta de uma instância na base conceitual com o predicado de seleção. Os nodos cinza pertencem aos fragmentos derivados. O fragmento horizontal corresponde ao grafo de dados selecionado.

A definição de fragmentação horizontal é dada abaixo. Primeiro, a função $getDataGraph()$ é definida. Esta função é responsável pela obtenção dos fragmentos derivados de uma dada instância conceitual. Esta função é especificada na Definição 4.6. A Definição 4.7 define o operador de fragmentação horizontal. Ele pega uma base conceitual e um parâmetro σ, contendo o predicado de seleção, e produz uma base conceitual fragmentada. A base conceitual fragmentada inclui todos os fragmentos primários e derivados.

DEFINIÇÃO 4.6 *A função* $getDataGraph(CM_B, i_R) \rightarrow CM'_B$ *retorna uma base conceitual contendo o grafo de dados no qual uma dada instância* i_r *pertence. Dados uma base conceitual* CM_B *e uma instância* $i_R \in (CM_B.L_B \cup CM_B.NL_B)$*, uma base conceitual* CM'_B *é mapeada como saída da função, tal que:*

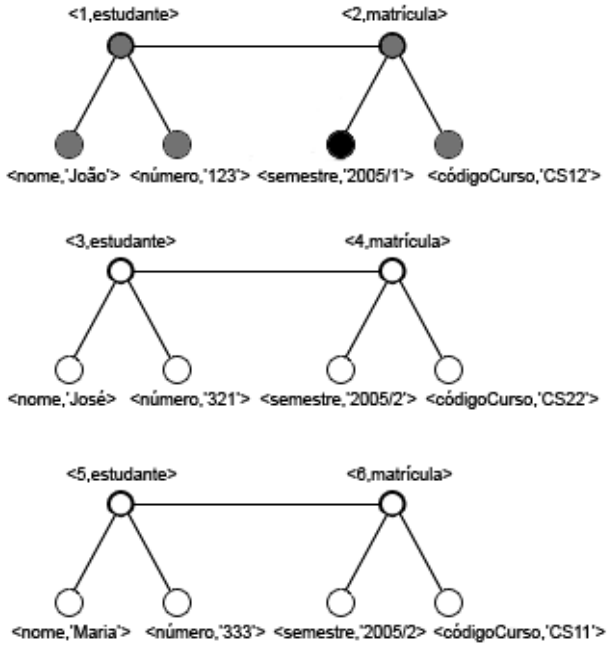

Figura 4.4: Exemplo de fragmentação horizontal

- *Para todas instâncias* $i_d \in (CM.L_B \cup CM.NL_B)$*, se existe um conjunto de instâncias* $\{i_1, i_2, \ldots, i_{n-1}\} \subseteq (CM.L_B \cup CM.NL_B)$ *e um conjunto de relacionamentos* $\{r_1, r_2, \ldots, r_n\}$*, onde* $r_1.i_1 = i_1$*,* $r_1.i_2 = i_R$*,* $r_2.i_1 = i_1$*,* $r_2.i_2 = i_2$*,* $r_3.i_1 = i_2$*,* $r_2.i_2 = i_3$*, ... ,* $r_n.i_1 = i_{n-1}$*,* $r_n.i_2 = i_d$*, então se* $i_d \in CM_B.NL_B$*, então* $i_d \in CM'_B.NL_B$*, senão* $i_d \in CM'_B.L_B$*.*

- *Para todos relacionamentos* $r \in CM_B.R_B$*, se* $r.I_1 \in CM'_B.NL_B$ *e* $r.I_2 \in (CM'_B.NL_B \cup CM'_B.NL_B)$*, então* $r \in CM'_B.R_B$*.*

Definição 4.7 *A função HF (Horizontal Fragmentation)* $HF(CM_B, \sigma) \rightarrow CM_{B1}$ *define o operador de fragmentação horizontal, onde* CM_B *é uma base conceitual do modelo conceitual* CM*,* σ *é o predicado de seleção e* CM_{B1} *é a base conceitual produzida como saída, tal que:*

- *Para cada instância* $i_1 \in CM_B.L_B$*, se* i_1 *obedece* σ*, então* $getDataGraph(CM_B, i_1) \subseteq CM_{B1}$*.*

4.5 Conclusão

Este Capítulo apresenta uma série de operadores de fragmentação, incluindo uma definição formal dos mesmos sobre o modelo de dados, o qual também é descrito formalmente no Capítulo anterior. Este modelo de dados inclui o modelo conceitual global, modelo conceitual local e suas respectivas bases conceituais.

O trabalho de formalização dos operadores de fragmentação e do modelo de dados foi realizado visando fornecer um arcabouço formal para o mecanismo de decomposição de consultas, o qual é detalhado no Capítulo 6. Porém, esta formalização também pode ser utilizada independentemente. Através dos operadores, é

possível descrever uma infinidade de possíveis cenários de integração de fontes XML. Definido-se regras de restrição, pode-se especificar claramente cenários consistentes, como por exemplo garantir que múltiplos esquemas conceituais compostos formem um esquema conceitual global, sem que haja perdas ou excesso de informação. Essas regras são conhecidas na literatura como regras de completeza, disjunção e reconstrução (OZSU; VALDURIEZ, 1991). A definição de tais regras está fora do escopo deste trabalho, e é sugerida como trabalho futuro.

5 MAPEAMENTO ENTRE O MODELO LOCAL E AS FONTES XML

Este Capítulo descreve a informação que é mantida relativa ao mapeamento entre um esquema local e o correspondente esquema local XML. Esta informação é usada pelo mecanismo de decomposição durante o processo de tradução de uma consulta global em uma consulta local. A abordagem de mapeamento utilizada é conhecida na literatura como *global-as-view* (ELMAGARMID; RUSINKIEWICZ; SHETH, 1999). Esta abordagem foi utilizada também em um trabalho anterior de nosso grupo de pesquisa (CAMILLO; MELLO; HEUSER, 2003), sendo estendida aqui com informações adicionais necessárias ao mecanismo de decomposição.

Dois tipos de mapeamento são definidos: *absoluto* e *relativo*. Mapeamentos absolutos são usados para descrever como um conceito no nível conceitual é encontrado no nível XML. Mapeamentos absolutos são descritos através de expressões XPath absolutas, uma para cada conceito e para cada fonte. Mapeamentos relativos são usados para descrever como um relacionamento de travessia (*traversal*) no nível conceitual é mapeado para uma navegação entre elementos no nível XML. Mapeamentos relativos são descritos através de expressões XPath relativas que navegam de um elemento para outro no nível XML.

Abaixo são apresentados alguns exemplos de informação de mapeamento para os esquemas conceituais locais da Figura 4.2, considerando a estrutura dos arquivos XML mostrada nas Figuras 5.1 e 5.2.

A Tabela 5.1 apresenta mapeamentos absolutos. Para cada conceito não-léxico em um dado esquema conceitual local e para cada fonte XML, haverá uma expressão XPath que retorna os elementos XML que representam aqueles conceitos. Por exemplo, considere o esquema local LCM_2 (Figura 4.2) e a fonte XML (*cursos.xml*) que

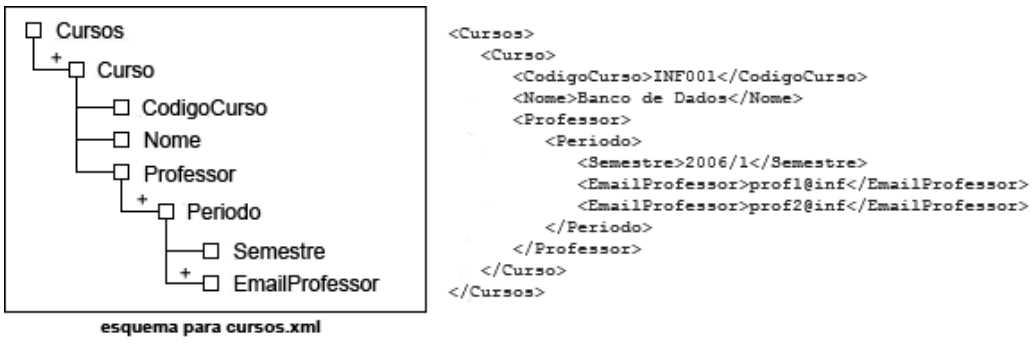

Figura 5.1: Esquema para as fontes XML do esquema LCM_2

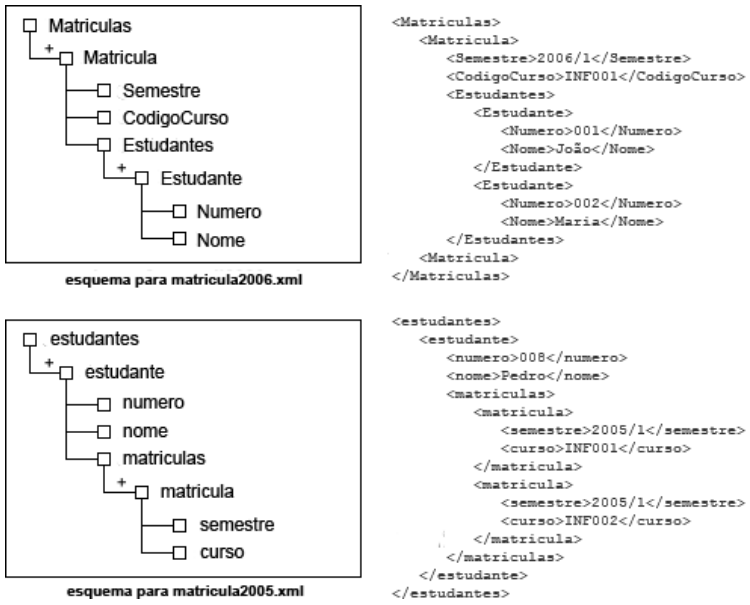

Figura 5.2: Esquema para as fontes XML do esquema LCM_3

Tabela 5.1: Informação de Mapeamento Absoluta

conceito	*LCM*	*url*	*expressão*
Curso	LCM_2	`cursos.xml`	`/Cursos/Curso`
Leciona	LCM_2	`cursos.xml`	`/Cursos/Curso/Professor/Periodo`
Matrícula	LCM_3	`matricula2005.xml`	`/estudantes/estudante/matriculas/matricula`
Matrícula	LCM_3	`matricula2006.xml`	`/Matriculas/Matricula`
Estudante	LCM_3	`matricula2005.xml`	`/estudantes/estudante`
Estudante	LCM_3	`matricula2006.xml`	`/Matriculas/Matricula/Estudantes/Estudante`

é relacionada ao mesmo (Figura 5.1). O esquema local LCM_2 contém dois conceitos não-léxicos: *Curso* e *Leciona*. O conceito *Curso* no nível conceitual é mapeado para a expressão XPath `/Cursos/Curso` no nível XML. Da mesma forma, o mapeamento do conceito Leciona para esta fonte é dado pela expressão XPath `/Cursos/Curso/Professor/Periodo`.

A Tabela 5.2 apresenta alguns exemplos de mapeamentos relativos. Para cada relacionamento de travessia no esquema conceitual local e para cada fonte XML, existe uma expressão XPath relativa que define como um relacionamento de travessia no nível conceitual é mapeado para o nível XML. Por exemplo, considere o relacionamento entre os conceitos *Curso* e *CódigoCurso* no esquema conceitual local LCM_2 (Figura 4.2). A travessia para este relacionamento de *Curso* para *CódigoCurso* é mapeado para a expressão XPath relativa `CodigoCurso`. Esta expressão especifica que, para navegar nesta fonte XML do elemento que corresponde ao conceito *Curso* ao elemento que corresponde ao conceito *CódigoCurso*, a expressão XPath relativa `CodigoCurso` deve ser aplicada. A expressão XPath relativa que mapeia a travessia do mesmo relacionamento na direção oposta (de CódigoCurso para Curso) é `..` (o operador para navegar para o elemento pai do XPath).

Uma definição formal para as informações de mapeamento é dada abaixo:

Definição 5.1 *Uma informação de mapeamento MI (Mapping Information) é uma tupla* $MI =< CM, LMI >$*, onde CM é um modelo conceitual global e LMI (Local Mapping Information) é um conjunto de informações de mapeamento locais. Uma informação de mapeamento local é uma tupla* $LMI =< LCM, AM, RM >$*, onde:*

Tabela 5.2: Informação de Mapeamento Relativa

origem	*destino*	*modelo*	*url*	*expressão*
Curso	CódigoCurso	LCM_2	`cursos.xml`	`CodigoCurso`
Curso	NomeCurso	LCM_2	`cursos.xml`	`Nome`
CódigoCurso	Curso	LCM_2	`cursos.xml`	`..`
NomeCurso	Curso	LCM_2	`cursos.xml`	`..`
Matrícula	Semestre	LCM_3	`matricula2005.xml`	`semestre`
Matrícula	CódigoCurso	LCM_3	`matricula2005.xml`	`curso`
Matrícula	Estudante	LCM_3	`matricula2005.xml`	`../..`
Semestre	Matrícula	LCM_3	`matricula2005.xml`	`..`
CódigoCurso	Matrícula	LCM_3	`matricula2005.xml`	`..`
Estudante	Matrícula	LCM_3	`matricula2005.xml`	`matriculas/matricula`
Estudante	Number	LCM_3	`matricula2005.xml`	`numero`
Estudante	Name	LCM_3	`matricula2005.xml`	`nome`
Número	Estudante	LCM_3	`matricula2005.xml`	`..`
Nome	Estudante	LCM_3	`matricula2005.xml`	`..`
Matrícula	Semestre	LCM_3	`matricula2006.xml`	`Semestre`
Matrícula	CódigoCurso	LCM_3	`matricula2006.xml`	`CodigoCurso`
Matrícula	Estudante	LCM_3	`matricula2006.xml`	`Estudantes/Estudante`
Semestre	Matrícula	LCM_3	`matricula2006.xml`	`..`
CódigoCurso	Matrícula	LCM_3	`matricula2006.xml`	`..`
Estudante	Matrícula	LCM_3	`matricula2006.xml`	`../..`
Estudante	Número	LCM_3	`matricula2006.xml`	`Numero`
Estudante	Nome	LCM_3	`matricula2006.xml`	`Nome`
Número	Estudante	LCM_3	`matricula2006.xml`	`..`
Nome	Estudante	LCM_3	`matricula2006.xml`	`..`

- *LCM é um modelo conceitual local de MI.CM.*
- *AM (Absolute Mappings) é um conjunto de informações de mapeamento absolutas. Uma informação de mapeamento absoluta é uma tupla* $AM =< c, url, expr >$*, onde* $c \in LCM.CM'.NL$*, url é a url da fonte e expr é a expressão XPath que mapeia o conceito c na fonte url.*
- *RM (Relative Mappings) é um conjunto de informações de mapeamento relativas. Uma informação de mapeamento relativa é uma tupla* $RM =< r, url, expr >$*, onde* $r \in LCM.CM'.R$*, url é a url da fonte e expr é a expressão XPath que mapeia o conceito c na fonte url.*

6 ALGORITMO DE DECOMPOSIÇÃO

Neste Capítulo, é descrito como uma consulta expressa no nível conceitual através de uma expressão CXPath (Subseção 2.3.2) é reescrita para uma expressão XQuery no nível XML. Mais especificamente, o algoritmo de decomposição inicialmente traduz a expressão CXPath em várias expressões XPath, uma para cada fonte XML, e a seguir agrupa essas subconsultas XPath em uma única expressão XQuery.

Os principais passos do algoritmo de decomposição são os seguintes:

1. **Executar o reconhecimento (*parse*) da consulta CXPath**
2. **Tratar os fragmentos horizontais**
3. **Navegar nos fragmentos**
4. **Efetuar a junção dos fragmentos split**
5. **Implementar os predicados de seleção**

O algoritmo de decomposição é explicado aqui discutindo os resultados de cada passo do algoritmo para um exemplo de consulta CXPath. O processo em alto nível é descrito no Algoritmo 1. Além disso, a versão do algoritmo explicada aqui não trata a fragmentação vertical. O objetivo desta omissão é simplificar a apresentação do algoritmo. As alterações necessárias para incluir este tipo de fragmentação são discutidas na Seção 6.7.

Nas Seções seguintes será mostrado como a expressão CXPath abaixo é traduzida quando executada contra o esquema conceitual global descrito na Figura 4.1 e os esquemas conceituais locais descritos na Figura 4.2, com as informações de mapeamento declaradas nas Tabelas 5.1 e 5.2 sendo consideradas.

```
/Curso[CódigoCurso="INF001"]/Matrícula[Semestre="2006/1"]
/Estudante/Nome
```

6.1 Reconhecimento da Consulta

O primeiro passo da decomposição é fazer o reconhecimento da consulta CXPath e gerar uma tupla para cada conceito da mesma. As subconsultas CXPath que aparecem nos predicados de seleção são tratadas através de chamadas recursivas deste algoritmo, conforme explicado abaixo. Cada tupla contém um identificador

Algoritmo 1: decomposeQuery

```
   input  : Uma consulta global CXPath, informações de mapeamento MI, o conceito associado bindedOid
            e bindedConcept
   output : Uma consulta local XQuery
 1 begin
 2     parsedQuery ← parseQuery(CXPath, MI, bindedOid, bindedConcept)
 3     if bindedConcept ≠ empty then
 4         binded = true
 5     else
 6         binded = false
 7     add result of handleHorizontalFragments(parsedQuery, MI, binded) into canonicalQuery.FOR
 8     add result of navigateInsideFragments(parsedQuery, MI) into canonicalQuery.FOR
 9     add result of joinSplitFragments(parsedQuery, MI) into canonicalQuery.WHERE
10     add result of handleSelectionPredicates(parsedQuery, MI) into canonicalQuery.WHERE
11     XQuery ← writeFinalQuery(canonicalQuery)
12 end
```

Tabela 6.1: Resultado do *parseQuery*

oid	*conceito*	*LCM*	*fontes*	*predicado*
1	Curso	LCM_2	`cursos.xml`	`CódigoCurso="INF001"`
2	Matrícula	LCM_3	`matricula2005.xml` `matricula2006.xml`	`Semestre="2006/1"`
3	Estudante	LCM_3	`matricula2005.xml` `matricula2006.xml`	
4	Nome	LCM_3	`matricula2005.xml` `matricula2006.xml`	

(*object id*), o conceito, o esquema conceitual local a que o conceito pertence, suas fontes, e o predicado de seleção associado a este conceito na expressão CXPath.

O identificador da tupla é usado durante a geração das cláusulas FOR da expressão XQuery resultante, para nomear cada variável com um $v seguido pelo *object id*. A Tabela 6.1 contém o resultado deste passo para a consulta de exemplo. A formalização do resultado gerado por este passo é dada na Definição 6.1.

DEFINIÇÃO 6.1 *O resultado do reconhecimento da consulta PQ (Parsed Query) é um conjunto de tuplas* $PQ =< oid, C, LCM, sources, predicate >$*, onde:*

- *oid é o object oid.*
- C *é um conceito, onde* $C \in MI.CM.NL \cup MI.CM.L$.
- LCM *é um modelo conceitual local, onde* $LCM \in MI.LMI.LCM$.
- *sources é um conjunto de url, onde* $url \in MI.LMI.AM.url$.
- *predicate é uma string.*

6.2 Tratar Fragmentos Horizontais

Neste passo, o algoritmo começa a construir a expressão XQuery. Para cada esquema conceitual local que é referenciado pelos conceitos existentes na expressão CXPath, uma associação (*binding*) de uma variável XQuery é construída. Essas associações serão usadas na cláusula FOR da expressão XQuery resultante. No caso deste exemplo, as associações descritas na Tabela 6.2 serão criadas.

Essas associações de variáveis são construídos como segue. Para cada esquema conceitual local, o algoritmo pega o primeiro conceito da parsedQuery (Tabela 6.1).

Tabela 6.2: Resultado de *handleHorizontalFragments* - cláusulas FOR

var	*expressão*
$v1	doc("cursos.xml")/Cursos/Curso
$v2	doc("matricula2005.xml")/estudantes/estudante/matriculas/matricula | doc("matriculas2006.xml")/Matriculas/Matricula

Neste exemplo, esses conceitos serão *Curso* no LCM_2 e *Matrícula* no LCM_3. Para cada conceito, uma associação é construída. Estas associações contém uma referência para cada fonte XML que é associada ao esquema local, assim como expressões de mapeamento absolutas (Seção 5) para o conceito em cada fonte, que especifica como as instâncias deste conceito são encontradas na fonte XML. Este processo é formalizado no Algoritmo 2.

O primeiro conceito do LCM_2 encontrado nesta consulta de exemplo é *Curso*. LCM_2 é representado pela fonte XML `cursos.xml` e a expressão de mapeamento absoluta para *Curso* em `cursos.xml` é `/Cursos/Curso`. Isto leva à primeira associação apresentada na Tabela 6.2. O esquema conceitual local LCM_3 é representado por duas fontes de dados XML diferentes, `matricula2005.xml` e `matricula2006.xml`. Neste caso, a associação irá corresponder a união dos elementos que representam o conceito *Matrícula* nessas fontes (segunda linha na Tabela 6.2).

Algoritmo 2: handleHorizontalFragments

```
   input  : Uma parsed query parsedQuery, informações de mapeamento MI e o parâmetro binded
   output : Um conjunto de cláusulas FOR
 1 begin
 2     LCM ← empty
 3     foreach PQ in parsedQuery do
 4         if (PQ.LCM ≠ LCM) AND (binded is not true) then
 5             counter ← 0
 6             expression ← empty
 7             foreach source in PQ.sources do
 8                 increment counter
 9                 if counter > 1 then
10                     expression ← expression + ' | '
11                 path ← findAbsolutePath(PQ.concept, source, MI)
12                 expression ← expression + 'doc("' + source + '")' + path
13             ForClause.var ← '$v' + PQ.oid
14             ForClause.expression ← expression
15             add ForClause into FOR
16         LCM ← PQ.LCM
17 end
```

6.3 Navegar nos Fragmentos

No segundo passo, foram definidas associações que são ligadas a expressões XPath absolutas e que serão usadas para iterar sobre cada fragmento, i.e., sobre cada esquema conceitual local. Neste passo, será implementada a navegação dentro de cada fragmento através de expressões XPath relativas. Para tratar este problema, será adotada a mesma abordagem descrita em (CAMILLO; MELLO; HEUSER, 2003).

O algoritmo compara cada par de conceitos adjacentes na parsedQuery. Se o par pertence ao mesmo esquema conceitual local, isto significa que um relacionamento entre os dois conceitos é um relacionamento de travessia (*traversal*). Para

Tabela 6.3: Resultado de *navigateInsideFragments* - cláusulas FOR

var	*expressão*
`$v3`	`$v2[base_uri(.)="matricula2005.xml"]/../..  \|` `$v2[base_uri(.)="matricula2006.xml"]/Estudantes/Estudante`
`$v4`	`$v3[base_uri(.)="matricula2005.xml"]/nome \|` `$v3[base_uri(.)="matricula2006.xml"]/Nome`

implementar este relacionamento de travessia no nível XML, a expressão XPath relativa definida por um mapeamento relativo para esta travessia (Seção 5) é ligada à variável. O Algoritmo 3 formaliza este processo. Por exemplo, considere o relacionamento do tipo travessia de *Matrícula* para *Estudante* no esquema local LCM_3. Esta travessia é implementada na fonte `matricula2005.xml` através da expressão XPath relativa `../..`, e na fonte `matricula2006.xml` através da expressão XPath relativa `Estudantes/Estudante` (veja a Tabela 5.2). Isto leva à primeira linha da Tabela 6.3.

Estas linhas definem uma associação para a nova variável `$v3`, a qual é ligada a uma expressão que é relativa a variável `$v2`. Recapitulando, a variável `$v2` é ligada a uma expressão XPath absoluta que recupera os elementos XML que representam o conceito *Matrícula*, fonte do relacionamento de travessia que está sendo tratado. A associação `$v3` construída neste passo especifica a travessia para o relacionamento de *Matrícula* para *Estudante* no nível conceitual. Como este relacionamento é implementado em duas fontes XML diferentes, os termos na forma `base_uri(.)` são utilizados para assegurar que esta expressão relativa é utilizada apenas sobre a fonte correta.

Analogamente, a variável `$v4` é ligada a expressão XPath relativa que especifica a travessia do conceito *Estudante* para o conceito *Nome*.

Algoritmo 3: navigateInsideFragments

```
   input  : Uma parsed query parsedQuery e informações de mapeamento MI
   output : Um conjunto de cláusulas FOR
 1 begin
 2     counter ← 0
 3     foreach PQ in parsedQuery do
 4         increment counter
 5         expression ← empty
 6         baseURI ← empty
 7         if counter = 1 then
 8             previousPQ ← PQ
 9             Next
10         if PQ.LCM = previousPQ.LCM then
11             counterSources ← 0
12             foreach source in PQ.sources do
13                 increment counterSources
14                 path ← findRelativePath(previousPQ.concept, PQ.concept, source, MI)
15                 if Count(PQ.sources) > 1 then
16                     baseURI ← '[base_uri(.)="' + source + ' "]'
17                 if counterSources > 1 then
18                     expression ← expression + ' | '
19                 expression ← expression + '($' + previousPQ.oid + baseURI + '/' + path + ')'
20             ForClause.var ← '$v' + PQ.oid
21             ForClause.expression ← expression
22             add ForClause into FOR
23         previousPQ ← PQ
24 end
```

Tabela 6.4: Resultado de *joinSplitFragments* - cláusulas WHERE

expressão
($v2[base_uri(.)="matricula2005.xml"]/curso | $v2[base_uri(.)="matricula2006.xml"]/CodigoCurso) = $v1/CodigoCurso

6.4 Efetuar Junção dos Fragmentos Split

Uma consulta CXPath pode acessar conceitos de diferentes esquemas locais, o que significa que diversos fragmentos split (esquemas locais) precisam ser acessados e integrados para responder a consulta. Para cada esquema conceitual local, a variável ligada a uma expressão XPath absoluta que recupera elementos para o fragmento foi construída no segundo passo do algoritmo. Neste passo, o critério para efetuar a junção dos elementos nas diferentes fontes é construído. Este critério irá fazer parte de uma cláusula WHERE na expressão XQuery resultante. O critério de junção será construído usando os *identificadores* e *identificadores de referência* que foram definidos durante a operação split que gerou os esquemas locais a partir do esquema global (Seção 3.1).

Na tabela parsedQuery (Tabela 6.1), para cada par adjacente de conceitos que pertencem a esquemas locais diferentes precisa ser definida uma junção. Para cada um desses pares de conceitos, o algoritmo encontra os identificadores e identificadores de referência que são usados para navegar de um fragmento para outro. No exemplo (Tabela 6.1), existe um único par de conceitos que corresponde a navegação de um fragmento para outro, especificamente a travessia do conceito *Curso* para *Matrícula*. Este relacionamento é implementado pelo identificador de referência *Matrícula.CódigoCurso* que referencia o identificador *Curso.CódigoCurso*.

A seguir, para cada fonte XML, o algoritmo pega a expressão de caminho relativa que implementa a travessia de *Curso* para o identificador *CódigoCurso* e de *Matrícula* para o identificador de referência *CódigoCurso*. Finalmente, um predicado de igualdade entre essas duas expressões XPath relativas é incluído na cláusula WHERE, conforme é mostrado na Tabela 6.4.

O Algoritmo 4 formaliza esse processo.

6.5 Tratar Predicados de Seleção

Este passo do algoritmo é responsável pelo tratamento dos predicados de seleção que podem ocorrer na consulta CXPath. Estes predicados são reescritos para termos que são incluídos nas cláusulas WHERE da consulta XQuery resultante. O algoritmo suporta uma forma simplificada de predicado de seleção, a qual tem a seguinte estrutura:

```
CXPathLeft comp [CXPathRight | constant]
```

onde *CXPathLeft* e *CXPathRight* são consultas CXPath, as quais podem ser expressões relativas ou absolutas, $comp \in \{=, \neq, <, >, \leqslant, \geqslant\}$ é um operador de comparação, e *constant* é uma string.

O primeiro passo do algoritmo para tratamento de predicados de seleção consiste em fazer o reconhecimento dos predicados na consulta global. Para o exemplo, o resultado é mostrado na Tabela 6.5.

As subconsultas CXPath que podem ocorrer no predicado de seleção são traduzidas através de uma chamada recursiva do algoritmo de decomposição. Por exemplo,

Algoritmo 4: joinSplitFragments

```
   input  : Uma parsed query parsedQuery e informações de mapeamento MI
   output : Um conjunto de cláusulas WHERE
 1 begin
 2     counter ← 0
 3     foreach PQ in parsedQuery do
 4         increment counter
 5         expressionLeft, expressionRight, baseURI ← empty
 6         if counter = 1 then
 7             previousPQ ← PQ
 8             Next
 9         if PQ.LCM ≠ previousPQ.LCM then
10             if getIDREF(previousPQ.concept, PQ.concept, MI) = empty then
11                 IDREFS ← getIDREF(PQ.concept, previousPQ.concept, MI)
12                 idrefConcept ← PQ.concept
13                 idrefConceptOid ← PQ.oid
14                 idrefSources ← PQ.sources
15                 idConcept ← previousPQ.concept
16                 idConceptOid ← previousPQ.oid
17                 idSources ← previousPQ.sources
18             else
19                 IDREFS ← getIDREF(previousPQ.concept, PQ.concept, MI)
20                 idrefConcept ← previousPQ.concept
21                 idrefConceptOid ← previousPQ.oid
22                 idrefSources ← previousPQ.sources
23                 idConcept ← PQ.concept
24                 idConceptOid ← PQ.oid
25                 idSources ← PQ.sources
26             foreach REF in IDREFS.REF do
27                 counterSources ← 0
28                 foreach source in idrefSources do
29                     increment counterSources
30                     path ← findRelativePath(idrefConcept, REF.ref.c2, source, MI)
31                     if Count(idrefSources) > 1 then
32                         baseURI ← = '[base_uri(.)="' + source + ' "]'
33                     if counterSources > 1 then
34                         expressionLeft ← expressionLeft + ' | '
35                     expressionLeft ← expressionLeft + '($v' + idrefConceptOid + baseURI + '/' + path
                       + ')'
36                 baseURI ← empty
37                 counterSources ← 0
38                 foreach source in idSources do
39                     increment counterSources
40                     path ← findRelativePath(idConcept, REF.id.c2, source, MI)
41                     if Count(idSources) > 1 then
42                         baseURI ← = '[base_uri(.)="' + source + ' "]'
43                     if counterSources > 1 then
44                         expressionRight ← expressionRight + ' | '
45                     expressionRight ← expressionRight + '($v' + idConceptOid + baseURI + '/' + path
                       + ')'
46                 WhereClause ← expressionLeft + '=' + expressionRight
47                 add WhereClause into WHERE
48         previousPQ ← PQ
49 end
```

Tabela 6.5: Resultado de *parsedSelectionPredicates*

bindedOid	*bindedConcept*	*predicado*	*CXPathLeft*	*CXPathRight*
1	Curso	`CódigoCurso="INF001"`	`CódigoCurso`	
2	Matrícula	`Semestre="2006/1"`	`Semestre`	

```
for
    $v1 in doc("cursos.xml")/Cursos/Curso,
    $v2 in doc("matricula2005.xml")/estudantes/estudante/matriculas/matricula
       | doc("matricula2006.xml")/Matriculas/Matricula,
    $v3 in $v2[base-uri(.)="matricula2005.xml"]/../.. |
           $v2[base-uri(.)="matricula2006.xml"]/Estudantes/Estudante,
    $v4 in $v3[base-uri(.)="matricula2005.xml"]/nome |
          $v3[base-uri(.)="matricula2006.xml"]/Nome
where
    ($v2[base-uri(.)="matricula2005.xml"]/curso |
     $v2[base-uri(.)="matricula2006.xml"]/CodigoCurso) =
     $v1/CodigoCurso and
    (for $v1-1 in $v1/CodigoCurso return $v1-1) = "INF001" and
    (for $v2-1 in $v2[base-uri(.)="matricula2005.xml"]/semestre |
         $v2[base-uri(.)="matricula2006.xml"]/Semestre
     return $v2-1) = "2006/1"
return
    $v4
```

Figura 6.1: XQuery gerada pelo algoritmo de decomposição para o exemplo 2.1

o predicado de seleção `CodigoCurso="INF001"` que contém uma expressão que é relativa ao conceito *Curso* é traduzida para o seguinte predicado:

```
(for $v1-1 in $v1/CodigoCurso
 return $v1-1)
 = "INF001"
```

A expressão XQuery `for $v1-1 ...` é obtida pela chamada recursiva do algoritmo de decomposição sobre a expressão CXPath relativa `CodigoCurso` que aparece no predicado de seleção. Quando expressões CXPath dentro de um predicado de seleção são relativas, o contexto, i.e. a tupla identificadora na parsedQuery e o conceito, precisam ser passados como parâmetro para o algoritmo de decomposição. Os detalhes referentes a essa chamada recursiva são detalhados na próxima subseção. O algoritmo que trata o reconhecimento dos predicados de seleção e as chamadas recursivas é formalizado no Algoritmo 5.

A consulta XQuery para o exemplo 2.1 é mostrada na Figura 6.1.

Algoritmo 5: handleSelectionPredicates

```
   input  : Uma parsed query parsedQuery, uma consulta canônica canonicalQuery e informações de
            mapeamento MI
   output : Um conjunto de cláusulas WHERE
 1 begin
 2     parsedSelectionPredicates ← parseSelectionPredicates(parsedQuery)
 3     foreach parsedPredicate in parsedSelectionPredicates do
 4         if NOT isAbsolute(parsedPredicate.CXPathLeft) then
 5             XQueryLeft ← decomposeQuery(CXPathLeft,MI,parsedPredicate.bindedOid,
               parsedPredicate.bindedConcept)
 6         else
 7             XQueryLeft ← decomposeQuery(CXPathLeft,MI,empty,empty)
 8         if parsedPredicate.CXPathRight ≠ empty then
 9             if NOT isAbsolute(parsedPredicate.CXPathRight) then
10                 XQueryRight ← decomposeQuery(CXPathRight,MI,parsedPredicate.bindedOid,
                   parsedPredicate.bindedConcept)
11             else
12                 XQueryRight ← decomposeQuery(CXPathRight,MI,empty,empty)
13         selectionPredicate ← parsedPredicate.predicate
14         selectionPredicate.replace(CXPathLeft, '(' + XQueryLeft + ')')
15         selectionPredicate.replace(CXPathRight, '(' + XQueryRight + ')')
16         add selectionPredicate into WHERE
17 end
```

Tabela 6.6: Resultado do reconhecimento da consulta em uma chamada recursiva

oid	*conceito*	*LCM*	*fontes*	*predicado*
1	Curso	LCM_2	`cursos.xml`	
1-1	CódigoCurso	LCM_2	`cursos.xml`	

DEFINIÇÃO 6.2 *O resultado do reconhecimento no predicado de seleção parsedSelectionPredicates é um conjunto de tuplas* $PSP =< bindedOid, bindedConcept, selectionPredicate, CXPathLeft, CXPathRight >$*, onde:*

- *bindedOid é o oid do conceito associado.*
- *bindedConcept é um conceito, onde* $bindedConcept \in MI.CM.NL \cup MI.CM.L$.
- *selectionPredicate é uma string contendo o predicado de seleção.*
- *CXPathLeft é uma consulta CXPath.*
- *CXPathRight é uma consulta CXPath.*

6.6 Tratamento de Expressões CXPath aninhadas

As expressões CXPath aninhadas (i.e. expressões CXPath dentro de predicados de seleção) são tratadas através de uma chamada recursiva ao algoritmo de decomposição. As execuções recursivas do algoritmo são similares à primeira iteração. Os passos que possuem um comportamento diferente são o de reconhecimento da consulta e o de tratamento de fragmentos horizontais. Os outros passos possuem sempre o mesmo comportamento, independente do nível de recursão. Nesta subseção estas diferenças serão explicadas através de uma execução detalhada da chamada recursiva feita para decompor a expressão CXPath relativa `CódigoCurso` dentro do predicado de seleção `CódigoCurso="INF001"` da consulta de exemplo.

Neste exemplo, a expressão `CódigoCurso` é relativa ao conceito ao qual o predicado de seleção é aplicado, que é o conceito `Curso`. Este conceito e o respectivo object id devem ser informados ao algoritmo de decomposição durante a chamada recursiva.

O passo para reconhecimento da consulta possui um comportamento diferente nas chamadas recursivas quando a consulta sendo decomposta é relativa. Ele deve incluir os conceitos associados na parsedQuery gerada, e mantém, para estes conceitos apenas, o *object id* utilizado nas iterações anteriores. O *object id* dos conceitos remanescentes serão gerados a partir da concatenação do *object id* do conceito ligado com um novo *object id*. No exemplo, a parsedQuery produzida é mostrada na Tabela 6.6.

O próximo passo do algoritmo é o tratar os fragmentos horizontais. O comportamento deste passo, quando se está executando a primeira iteração, é gerar uma associação para cada esquema conceitual local existente na consulta, conforme explicado na Seção 6.2. Na chamada recursiva para decompor uma expressão relativa, o primeiro esquema conceitual local na parsedQuery é ignorado. Isto é feito porque o algoritmo irá utilizar a associação gerada na recursão anterior, ao invés de declarar uma nova. No exemplo, nenhuma associação é gerada neste passo.

O passo para navegação nos fragmentos terá o mesmo comportamento em todos os níveis de recursão. No exemplo, este passo irá gerar as associações mostradas na

Tabela 6.7: Resultado de *navigateInsideFragments* na chamada recursiva - cláusulas FOR

var	*expressão*
`$v1-1`	`$v1/CodigoCurso`

Tabela 6.8: Resultado de *parseQuery* - modelos verticalmente fragmentados

oid	*conceito*	*LCM*	*fontes*	*predicado*
1	Leciona	LCM_2	`cursos.xml`	`Período="2001"`
2	Professor	LCM_{V1}	`prof1.xml`	`Departamento="Banco de Dados"`
3	Professor	LCM_{V2}	`prof2.xml`	`Departamento="Banco de Dados"`
4	Nome	LCM_{V2}	`prof2.xml`	

Tabela 6.7. Considere que esta é uma expressão de caminho relativa aplicada sobre uma associação declarada em um nível de recursão anterior do algoritmo.

Os passos restantes não irão produzir nenhum resultado para a consulta exemplo. A consulta resultante na chamada recursiva é a seguinte:

```
for $v1-1 in $v1/CodigoCurso return $v1-1
```

6.7 Tratamento de Fragmentação Vertical

Para simplificar a apresentação do algoritmo, o tratamento da fragmentação vertical não foi incluído nos passos anteriores. Para tratar este tipo de fragmentação, é necessário alterar vários dos passos apresentados anteriormente, acrescentando uma complexidade desnecessária e dificultando o entendimento do processo. Nesta Seção, são discutidas as alterações necessárias no algoritmo de decomposição, de modo que a fragmentação vertical seja endereçada. Por fim, é feita uma descrição informal do passo para tratamento da fragmentação vertical. Isto será explicado através da decomposição da seguinte consulta CXPath, executada contra o modelo conceitual da Figura 4.1 e os modelos conceituais locais LCM_2, LCM_{V1} e LCM_{V2}, descritos nas Figuras 4.2 e 4.3, respectivamente.

```
/Leciona[Período="2001"]/Professor[Departamento="Banco de Dados"]/Nome
```

A primeira alteração que precisa ser feita é no algoritmo responsável pelo reconhecimento da consulta. Para tratar fragmentos verticais, este algoritmo deve incluir na parsedQuery uma tupla para cada esquema conceitual local e para cada conceito da consulta sendo decomposta, ao invés de apenas uma única tupla para cada conceito. Com esta mudança, cada conceito verticalmente fragmentado irá possuir diversas tuplas. Neste exemplo, o conceito *Professor*, o qual é representado nos esquemas locais LCM_{V1} e LCM_{V2}, irá possuir duas tuplas na parsedQuery, como é mostrado na tabela 6.8.

O algoritmo Tratar Fragmentos Horizontais não precisa ter seu comportamento alterado, com a exceção de que um mesmo conceito poderá possuir mais que uma associação, devido à alteração no algoritmo de reconhecimento da consulta. As associações de variáveis FOR geradas para este exemplo estão mostradas na Tabela 6.9.

O algoritmo Navegar nos Fragmentos terá um comportamento diferente. Ao invés de comparar cada par de conceitos adjacentes na parsedQuery, este algoritmo deverá comparar todas as tuplas de um dado conceito com todas as tuplas do conceito adjacente. As tuplas para um mesmo conceito não deverão ser comparadas

Tabela 6.9: Resultado de *handleHorizontalFragments* - modelos verticalmente fragmentados

var	*expressão*
$v1	doc("cursos.xml")/Cursos/Curso
$v2	doc("prof1.xml")/Professor
$v3	doc("prof2.xml")/professores/professor

Tabela 6.10: Resultado de *navigateInsideFragments* - esquemas verticalmente fragmentados

var	*expressão*
$v4	$v3/nome

entre elas mesmas. No exemplo, a tupla 1 deve ser comparada com as tuplas 2 e 3 (navegação de *Leciona* para *Professor*), e as tuplas 2 e 3 devem ser comparadas com a tupla 4 (navegação de *Professor* para *Nome*). As tuplas que pertencem a um mesmo esquema local são aquelas as quais possuem um relacionamento de travessia. No exemplo, a navegação para o conceito *Nome* será implementada com uma expressão de caminho relativa sobre a associação $v3, que é a associação a qual pertence ao mesmo esquema conceitual local que a tupla 4, que é o LCM_{V2}. A cláusula FOR gerada é mostrada na Tabela 6.10.

O algoritmo Efetuar Junção de Fragmentos terá uma alteração similar no comportamento. Assim como no algoritmo Navegar nos Fragmentos, ele também irá comparar todas as tuplas de um dado conceito com todas as tuplas do conceito adjacente na parsedQuery. Se todos os esquemas locais forem diferentes, uma junção precisa ser implementada. No exemplo, isto é implementado entre as tuplas 1 e 2. O resultado deste passo é mostrado na Tabela 6.11.

O algoritmo Tratar Predicados de Seleção precisa de uma alteração em seu comportamento. Quando o algoritmo de decomposição é chamado recursivamente para uma expressão relativa, o contexto (conceito e oid associado ao predicado de seleção) deve ser passado como parâmetro, conforme explicado na Seção 6.5. Se o conceito relacionado possui fragmentação vertical, é necessário descobrir qual oid deve ser associado com aquela navegação. Isto é feito executando o reconhecimento na expressão CXPath relativa e checando se o primeiro conceito naquela expressão é representado no mesmo esquema conceitual local que um dos oids dos conceito relacionado. Se isto é verdadeiro, o oid relacionado com o mesmo esquema conceitual local que o primeiro conceito na expressão relativa é passado como parâmetro. Caso contrário, qualquer oid pode ser utilizado. Neste caso, o algoritmo pega o primeiro oid disponível. No exemplo, a expressão `Departamento`, relativa ao conceito *Professor*, é representada no esquema conceitual local LCM_{V1}. O oid 2 da parsedQuery representa o conceito *Professor* no esquema conceitual local LCM_{V1}, então este oid é passado como parâmetro para o algoritmo de decomposição. O resultado da decomposição é mostrado abaixo.

```
for $v2-1 in $v2/Departamento return $v2-1
```

Tabela 6.11: Resultado de *joinSplitFragments* - esquemas verticalmente fragmentados

expressão
($v1/ProfessorEmail = $v2/Email

Tabela 6.12: Resultado de *handleVerticalFragments* - cláusulas WHERE

expressão
`($v2/Email = $v3/email`

O algoritmo Tratar Fragmentação Vertical deve ser incluído como um dos passos do algoritmo de decomposição. Este passo terá um comportamento similar ao algoritmo Efetuar Junção dos Fragmentos Split. Ele será responsável pela geração do critério para junções dos diferentes fragmentos verticais. Assim como no algoritmo para efetuar junções, este critério fará parte da cláusula WHERE, usando o *identificador* do conceito fragmentado.

O comportamento deste algoritmo é o seguinte. Para cada par adjacente de tuplas na parsedQuery com o mesmo conceito, uma junção precisa ser definida. O algoritmo pega os identificadores deste conceito e, para cada fonte XML, pega a expressão de caminho relativa que implementa a travessia do conceito fragmentado para cada um dos identificadores. Então, um predicado de igualdade entre as duas associações é incluído na cláusula WHERE. No exemplo, existe um par adjacente de tuplas na parsedQuery com o mesmo conceito, que é o par formado pelas tuplas 2 e 3 para o conceito *Professor*. O identificador para o conceito *Professor* é o relacionamento com o conceito *ProfessorEmail*, então o próximo passo do algoritmo é pegar as expressões de caminho relativas entre esses conceitos, para cada fonte XML e para cada associação. Na fonte `prof1.xml` da tupla 2, este relacionamento é implementado com a expressão de caminho relativa `Email`, enquanto que na fonte `prof2.xml` da tupla 3, este relacionamento é implementado com a expressão `email`. A cláusula WHERE gerada é mostrada na Tabela 6.12.

7 CONCLUSÃO

Este trabalho apresenta uma abordagem para tratar o problema de efetuar consultas sobre fontes XML integradas. As principais contribuições deste trabalho são resumidas a seguir:

- Uma abordagem para fragmentação de documentos XML que pertencem a um domínio específico e cujos esquemas são integrados através de um modelo conceitual. Esta abordagem tem a característica de definir os operadores de fragmentação sobre um esquema global do tipo grafo, ao invés de definir os operadores sobre o nível hierárquico XML.

- A formalização desses operadores de fragmentação, fornecendo o arcabouço formal necessário para tratar adequadamente o problema da decomposição de consultas.

- Um algoritmo para decomposição de consultas escritas em CXPath contra o modelo conceitual. Este algoritmo tem a característica de se basear na fragmentação das fontes XML, conforme descrita através de operadores especificamente projetados para esta finalidade.

Uma das mais importantes contribuições deste trabalho é claramente separar o problema da fragmentação de um esquema global em vários esquemas locais do problema de abstrair diferentes representações XML de uma mesma informação conceitual. Esta separação é realizada através da introdução de uma camada conceitual no nível local das fontes e também no nível global (mediado). Os operadores de fragmentação são definidos neste nível conceitual. Outra contribuição relevante é o algoritmo para decomposição de consultas, que generaliza os algoritmos para redução de fragmentação de bases de dados relacionais para o modelo de dados apresentado neste trabalho. São apresentadas definições formais para os modelos de dados e operadores de fragmentação, e algoritmos que referenciam explicitamente essas definições.

Todavia, alguns dos problemas relacionados ao mecanismo de decomposição de consultas continuam em aberto. O problema de integração de instâncias, i.e. como identificar e integrar uma mesma instância que esteja representada em diferentes fontes deve ser investigado. Por exemplo, duas fontes contendo informações sobre estudantes podem conter diferentes instâncias relativas a um mesmo aluno. Para tratar esse problema, é possível seguir dois caminhos: (1) definir regras de completeza, reconstrução e disjunção para os operadores de fragmentação, as quais garantiriam um estado consistente para a base distribuída e (2) tornar o algoritmo

mais robusto, de modo a conseguir reconhecer e integrar as diferentes instâncias. Ambos caminhos não são mutuamente exclusivos, e podem tratar diferentes classes de problemas.

Outro ponto que não é tratado neste trabalho é o da otimização de consultas. As consultas geradas por este algoritmo não são as mais eficientes, e na maioria das vezes a performance da execução pode ser melhorada simplesmente reescrevendo essas consultas. Uma investigação futura poderá decidir se a melhor alternativa para tratar o problema da otimização é melhorando a consulta XQuery gerada ou utilizando alguma álgebra XML (JAGADISH et al., 2002; ZHANG; RUNDENSTEINER, 2002), após o processo de decomposição ter sido concluído.

REFERÊNCIAS

ABITEBOUL, S. et al. Dynamic XML documents with distribution and replication. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, SIGMOD, 2003, New York. **Proceedings...** New York: ACM Press, 2003. p.527–538.

BOSE, S. et al. A Query Algebra for Fragmented XML Stream Data. In: DBPL, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.195–215.

BREMER, J.-M.; GERTZ, M. On Distributing XML Repositories. In: WEBDB, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.73–78.

CAMILLO, S.; MELLO, R. S.; HEUSER, C. A. Querying Heterogeneous XML Sources through a Conceptual Schema. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 22., 2003, Chicago. **Proceedings...** Berlin: Springer-Verlag, 2003. p.186–199.

CERI, S.; NEGRI, M.; PELAGATTI, G. Horizontal data partitioning in database design. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, SIGMOD, 1982, New York. **Proceedings...** New York: ACM Press, 1982. p.128–136.

CERI, S.; PELAGATTI, G. Correctness of Query Execution Strategies in Distributed Databases. **ACM Trans. Database Syst.**, [S.l.], v.8, n.4, p.577–607, 1983.

DOAN, A.; DOMINGOS, P.; HALEVY, A. Y. Reconciling Schemas of Disparate Data Sources: a machine-learning approach. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, SIGMOD, 2001. **Proceedings...** New York: ACM Press, 2001.

ELMAGARMID, A.; RUSINKIEWICZ, M.; SHETH, A. (Ed.). **Management of heterogeneous and autonomous database systems**. San Francisco, CA, USA: Morgan Kaufmann, 1999.

HALPHIN, T. **Object-Role Modeling (ORM/NIAM). Handbook on Architectures of Information Systems**. [S.l.]: Springer-Verlag, 1998. 81-102p.

JAGADISH, H. V. et al. TAX: a tree algebra for xml. In: INTERNATIONAL WORKSHOP ON DATABASE PROGRAMMING LANGUAGES, 8., 2002. **Revised Papers...** Berlin: Springer-Verlag, 2002. p.149–164.

JOSIFOVSKI, V.; RISCH, T. Query Decomposition for a Distributed Object-Oriented Mediator System. **Distributed and Parallel Databases**, [S.l.], v.11, n.3, p.307–336, 2002.

LYNGBAEK, P. **OSQL**: a language for object databases. [S.l.]: HP Labs, 1991.

MA, H. et al. Distribution Design for XML Documents. In: ICECE, 2003. **Proceedings...** [S.l.: s.n.], 2003.

MA, H.; SCHEWE, K.-D. Fragmentation of XML Documents. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, SBBD, 2003. **Anais...** [S.l.: s.n.], 2003. p.200–214.

MACKINNON, L. M.; MARWICK, D. H.; WILLIAMS, M. H. A Model for Query Decomposition and Answer Construction in Heterogeneous Distributed Database Systems. **J. Intell. Inf. Syst.**, [S.l.], v.11, n.1, p.69–87, 1998.

MANOLESCU, I.; FLORESCU, D.; KOSSMANN, D. Answering XML Queries on Heterogeneous Data Sources. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 27., 2001. **Proceedings...** San Francisco: Morgan Kaufmann, 2001. p.241–250.

MELLO, R. S.; CASTANO, S.; HEUSER, C. A. A Method for the Unification of XML Schemata. **Information and Software Technology**, Amsterdam, v.44, n.4, p.241–249, 2002.

MELLO, R. S.; HEUSER, C. A. A Rule-Based Conversion of a DTD to a Conceptual Schema. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 20., 2001, Yokohama, Japan. **Proceedings...** Berlin: Springer-Verlag, 2001. p.133–148.

MELLO, R. S.; HEUSER, C. A. BInXS: a process for integration of xml schemata. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAISE, 17., 2005, Porto, Portugal. **Proceedings...** Berlin: Springer-Verlag, 2005. p.151–166.

NAVATHE, S. et al. Vertical partitioning algorithms for database design. **ACM Trans. Database Syst.**, New York, NY, USA, v.9, n.4, p.680–710, 1984.

OZSU, M. T.; VALDURIEZ, P. **Principles of distributed database systems**. Upper Saddle River, NJ, USA: Prentice-Hall, 1991.

REYNAUD, C.; SIROT, J.-P.; VODISLAV, D. Semantic Integration of XML Heterogeneous Data Sources. In: IDEAS, 2001. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.199–208.

SATTLER, K.-U.; GEIST, I.; SCHALLEHN, E. Concept-based querying in mediator systems. **The VLDB Journal**, Secaucus, NJ, USA, v.14, n.1, p.97–111, 2005.

SCHEWE, K.-D. Fragmentation of Object Oriented and Semistructured Data. In: BALTIC CONFERENCE, BALTICDB&IS, 2002. **Proceedings...** Tallinn: Institute of Cybernetics at Tallinn Technical University, 2002. p.253–266.

SHIPMAN, D. W. The Functional Data Model and the Data Language DAPLEX. **ACM Trans. Database Syst.**, [S.l.], v.6, n.1, p.140–173, 1981.

SUCIU, D. Distributed query evaluation on semistructured data. **ACM Transactions on Database Systems**, [S.l.], v.27, n.1, p.1–62, 2002.

TAMHANKAR, A. M.; RAM, S. Database fragmentation and allocation: an integrated methodology and case study. **IEEE Transactions on Systems, Man, and Cybernetics, Part A**, [S.l.], v.28, n.3, p.288–305, 1998.

W3C. Disponível em: $< http://www.w3.org/ >$. Acesso em: out. 2006.

WIEDERHOLD, G. Mediators in the Architecture of Future Information Systems. **Computer**, Los Alamitos, CA, USA, v.25, n.3, p.38–49, 1992.

ZHANG, X.; RUNDENSTEINER, E. **XAT XML Algebra for Rainbow system**. [S.l.]: Worcester: Computer Science Department, WPI, 2002. (Technical Report WPI- CS-TR-02-24).