Integração de Fontes de Dados Heterogêneas Baseadas em Ambientes Flexíveis e Dinâmicos

Angelo Brayner
Universidade de Fortaleza (UNIFOR), Departamento de Ciência da Computação,
Fortaleza, Brasil, 60811-341,
brayner@unifor.br

e

Marcelo Meirelles
Universidade de Fortaleza (UNIFOR), Departamento de Ciência da Computação,
Fortaleza, Brasil, 60811-341,
meirelles.mia@unifor.br

Resumo
Ambientes flexíveis e dinâmicos são caracterizados pela alta independência dos participantes da conexão, pelo baixo controle sobre os serviços solicitados e disponibilizados e pela necessidade de uma alta tolerância às falhas de comunicação. Integrar fontes de dados baseadas nesses ambientes requer uma estratégia de integração que garanta uma maior autonomia para as fontes de dados locais. Por isso, esse trabalho propõe a utilização da arquitetura **MDBS** (Multidatabase System) para integrar fontes de dados heterogêneas disponibilizadas em ambientes flexíveis e dinâmicos. Na arquitetura MDBS, a linguagem de consultas é responsável por mapear e resolver todos os conflitos de integração e, portanto, deve possuir instruções especiais que permitam identificar tais conflitos. Conseqüentemente, esse artigo propõe, ainda, uma extensão à linguagem XQuery, denominada **MXQuery**, que apresenta suporte necessário à especificação de consultas que acessam múltiplas fontes de dados heterogêneas e distribuídas baseadas em um modelo de dados XML. Assim, a MXQuery pode ser incorporado a um MDBS para integrar fontes de dados heterogêneas.
Palavras chaves: Banco de Dados, Banco de Dados Múltiplos, Integração de Fontes de Dados, Fontes de Dados Heterogêneas, Linguagens de Consulta.

Abstract
Flexible and dynamic environments are characterized by high independence from connection participants, low control over available services and high tolerance to communication failures. Integrating data sources published on such environments requires an integration strategy that guarantees autonomy to the local data sources. Multidatabase Systems (**MDBS**) has been consolidated as an approach to integrate multiple heterogeneous and distributed data sources. A key property of MDBSs is to guarantee a higher autonomy to the local data sources than the other approaches for integrating heterogeneous data sources. MDBS technology uses a query language as integration mechanism, which is responsible for solving the integration conflicts. Thus, the query language must provide constructs to perform queries over several different data sources and capable of solving integration conflicts. This paper proposes an extension to the XQuery language, called **MXQuery**. The key feature of the proposed language is to provide mechanisms, which support the capability to jointly manipulate data in different data sources based on an XML data model.
Keywords: Databases, Multidatabases, Data Sources Integration, Heterogeneous Data Sources, Query Languages.

1. Introdução

Cada vez mais, a **Web** (*World Wide Web*) tem sido utilizada como um ambiente para disponibilizar bancos de dados, porém utilizar essa arquitetura impõe ao usuário as vantagens e desvantagens desse ambiente. A Web pode ser vista como uma grande rede constituída a partir da união de várias redes locais (**LAN** – *Local Area Network*) espalhadas pelo mundo inteiro, dessa forma, é possível acessar um banco de dados localizado em uma rede local distinta através da malha de rede formada pelo ambiente Web. Contudo, as redes locais, que integram a Web, são independentes entre si, podendo conectar-se ou desconectar-se da rede mundial a qualquer momento.

Recentemente, alguns trabalhos têm abordado a conexão de redes locais sem fio e de forma dinâmica, essas redes são conhecidas como **MANET** (*Mobile Ad hoc Networking*) [4]. As redes MANET possuem um raio de cobertura e os equipamentos que entram nesse raio de cobertura passam automaticamente a fazer parte da rede estabelecida. Portanto, equipamentos móveis podem trafegar por diversas áreas de cobertura de redes MANET distintas e, embora, esse ambiente esteja limitado a uma pequena área de abrangência, ele possui algumas características semelhantes ao ambiente Web, tais como: alta independência dos participantes da conexão; baixo controle sobre os serviços solicitados e disponibilizados; alta tolerância às falhas de comunicação.

Nesse sentido, pode-se vislumbrar um cenário, onde vários bancos de dados podem estar inteconectados através de um sistema de comunicação sem fio. Dessa forma, um usuário, a partir de um computador móvel qualquer, poderia acessar quaisquer bancos de dados, residindo também em computadores móveis, desde que os computadores móveis estivessem localizados em uma região coberta por um sistema de comunicação sem fio. Uma coleção de computadores móveis interconectados por uma infra-estrutura de comunicação sem fio é denominada de comunidade de bancos de dados móveis (**MDBC** – *Mobile Database Comunity*), onde cada participante desta comunidade pode atuar como um servidor de banco de dados autônomo [4].

Portanto, integrar bancos de dados disponibilizados na Web e em redes MANET tem se tornado um grande desafio para área de banco de dados. Além disso, deve-se considerar a possibilidade desses bancos de dados serem heterogêneos, onde, nesse caso, para fornecer uma resposta consolidada ao usuário, é necessário eliminar as diferenças existentes (resolução de conflitos) entre os bancos de dados heterogêneos participantes. Várias abordagens têm sido propostas para viabilizar essa integração, cada uma delas sendo mais adequada a uma determinada situação. Por esse motivo, a definição da estratégia de integração mais apropriada deve considerar as características dos bancos de dados a serem integrados e o contexto em que a integração é necessária.

Uma abordagem para integração de bancos de dados, que estejam baseados na Web ou em redes *ad hoc*, deve oferecer suporte às características desses ambientes. Portanto, a arquitetura de integração deve conferir um alto grau de autonomia para os participantes. Além disso, deve ser capaz de resolver conflitos de integração originados, principalmente, da grande heterogeneidade dos bancos de dados a serem integrados. Finalmente, deve permitir uma manipulação de dados distribuídos em diferentes locais de forma eficiente.

Uma estratégia para integrar múltiplos bancos de dados heterogêneos é a de Sistema de Bancos de Dados Múltiplos (**MDBS** – *Multidatabase System*). A tecnologia de MDBS garante um maior grau de autonomia para os bancos de dados participantes da integração, denominados de Sistemas de Bancos de Dados Locais (**LDBS** – *Local Database System*) [8, 16, 6]. Entretanto, utilizar essa abordagem obriga o usuário a definir questões de localização e resolução de problemas de integração, como, por exemplo, conflitos semânticos e estruturais entre os dados armazenados nos diversos LDBS's. Em um MDBS, não é necessária a definição de um esquema global de integração, pois a linguagem de consulta (**MDL** – *Multidatabase Language*) é utilizada como mecanismo de integração. A idéia é que os usuários sejam capazes de "enxergar" cada esquema dos LDBS's representados em um modelo comum de dados (**CDM** – *Common Data Model*) [8]. Os esquemas locais representados nesse modelo comum são denominados de esquema conceitual.

A idéia básica desse trabalho é utilizar o **XML** (*eXtended Markup Language*) como modelo comum (esquema conceitual) dos dados armazenados nas múltiplas fontes de dados heterogêneas. Conseqüentemente, para adotar a arquitetura MDBS, deve-se utilizar uma linguagem de consultas MDL que seja capaz de manipular bancos de dados múltiplos de forma integrada e baseada em um modelo de dados XML. Assim, foi proposta a linguagem MXQuery que é uma extensão da linguagem XQuery (*Working-in-progress*). A MXQuery resolve problemas de integração de dados como heterogeneidade semântica e o tratamento de informações incompletas. A estrutura da extensão é bastante flexível, pois garante a integração de um número variável de fontes de dados com diferentes graus de autonomia.

Além disso, esse trabalho apresenta uma arquitetura para o processamento de consultas em sistemas de banco de dados múltiplos. A idéia é que a linguagem MXQuery seja incorporada a esta arquitetura como linguagem de banco de dados múltiplos.

O resto desse artigo está organizado como se segue. Uma breve apresentação de trabalhos relacionados é feita na seção 2. A proposta de extensão a linguagem XQuery é apresentada na seção 3. A seção 4 apresenta mais detalhadamente as características da estratégia de integração proposta. A seção 5 discute o processamento de consultas MXQuery. Conclusões e trabalhos futuros aparecem na seção 6.

2. Trabalhos Relacionados

Inúmeros trabalhos apresentam linguagens de consultas para bancos de dados onde é possível manipular fontes de dados distintas. Em [12, 10], é apresentada uma linguagem MDL para manipulação de bancos de dados múltiplos. A linguagem MSQL possui mecanismos para integrar bancos de dados relacionais, que aderem ao padrão SQL de forma a permitir junções entre tais bancos de dados. Embora essa proposta confira maior expressividade à linguagem SQL, ela é limitada no tratamento de incompatibilidades estruturais. Apenas conflitos de nome (sinônimos e homônimos) são resolvidos através da utilização de variáveis semânticas e a junção entre bancos de dados é limitada a operações com atributos de domínios compatíveis [14]. Além disso, Krishnamurthy et al. apontam a incapacidade da linguagem MSQL em resolver discrepâncias esquemáticas ou conflitos de esquema [11].

Outro fator limitante da linguagem MSQL é permitir apenas a integração de esquemas relacionais. Conseqüentemente, para utilizar essa linguagem, seria necessário adotar como CDM o modelo de dados relacional. A tarefa de mapear modelos de dados pós-relacionais, em especial os baseados em dados semi-estruturados, para um modelo de dados tão rígido quanto o modelo de dados relacional é uma tarefa bastante complexa. Portanto, a MDL MSQL não é indicada para integrar fontes de dados baseadas na Web ou em redes *ad hoc*.

A proposta apresentada por Missier e Rusinkiewiez define atributos globais, que representarão o resultado integrado de uma consulta [14]. Em seguida, utilizam instruções declarativas de mapeamento entre os atributos dos bancos de dados locais e os atributos globais. Após a definição dos mapeamentos ou contexto de uma consulta, as consultas, que são expressas em uma linguagem derivada do SQL, podem ser submetidas. A utilização de junções implícitas possibilita expressar consultas mais flexíveis e orientadas ao resultado.

A linguagem IDL apresentada por Krishnamurthy et al. permite endereçar os problemas abordados pela linguagem MSQL, porém a principal característica da linguagem IDL é resolver problemas de conflito de esquemas, denominados de discrepâncias esquemáticas [11]. Embora parte desses conflitos possa ser resolvida através da linguagem MSQL, alguns problemas não encontram em MSQL uma solução, como, por exemplo, integrar informações representadas como dado em um esquema e metadado em outro esquema. Tanto a proposta apresentada por Krishnamurthy et al. [11] quanto à apresentada por Missier e Rusinkiewiez [14] possuem mais expressividade comparada à linguagem MSQL, contudo continuam limitadas pelo modelo de dados relacional.

Domenig e Dittrich apresentaram uma linguagem baseada em OQL, onde os dados podem ser visualizados em diferentes níveis de visões conceituais. Tais níveis de visões conceituais organizam os dados de fontes de dados distintas de acordo com características comuns. Esse tipo de apresentação foi chamado pelos autores de *data space*, que é utilizado para integrar dados estruturados, semi-estruturados e não estruturados. Apesar da linguagem OQL apresentar mecanismos para consultar conjuntos de dados, esses mecanismos não podem ser utilizados para consultar dados não estruturados e semi-estruturados. Por esse motivo, os autores apresentaram uma extensão à linguagem OQL, denominada SOQL [6].

A abordagem proposta pelos autores define duas etapas distintas para integração de fontes de dados. Na etapa de pré-integração, os administradores identificam similaridades e conflitos entre as fontes de dados e, em seguida, definem regras de correspondência entre esquemas (contexto da consulta). Na etapa de integração, o esquema integrado é automaticamente construído com base nas regras previamente estabelecidas. Embora a abordagem proposta pelos autores ofereça um certo grau de flexibilidade com a utilização das regras de correspondência, esse processo tem que ser executado antes da proposição de consultas. Além disso, é necessário redefinir o esquema integrado cada vez que uma regra é modificada. Caso a quantidade de regras de correspondência seja muito grande pode degradar a performance do sistema.

A proposta apresentada por Sattler et al. assemelha-se à estratégia adotada por [14]. A linguagem FRAQL possui instruções declarativas que possibilitam mapear os objetos de um esquema local em objetos de um esquema global [18]. Inicialmente, o esquema global, que estará acessível às consultas dos usuários, é definido. Em seguida, os esquemas locais são mapeados segundo o esquema global, onde é possível utilizar funções de conversão criadas pelos próprios usuários. As correspondências criadas pelo usuário ou pelo administrador formam o contexto de execução de consultas. As consultas são submetidas considerando o esquema global e, em seguida, são transformadas em sub-consultas utilizando os mesmos critérios estabelecidos no mapeamento.

Diferentemente da proposta adotada por [6], essa abordagem não necessita que o esquema global seja refeito cada vez que uma nova fonte de dados entre na comunidade, sendo necessário, apenas, que essa nova fonte de dados estabeleça sua correspondência com o esquema já definido. Embora essa proposta seja mais flexível, o fato de a etapa de mapeamento estar dissociada da consulta propriamente dita, torna a especificação de consultas menos dinâmica que a abordagem utilizada nesse artigo. Além disso, a abordagem proposta em [18] está direcionada ao modelo de dados objeto-relacional e, embora seja mais flexível que o modelo relacional para representar dados semi-estruturados, ainda necessita de algum esforço para representar fontes de dados semi-estruturadas e não estruturadas.

A linguagem XQuery [5, 17, 2] possui várias características que permitem consultar documentos XML e apresentar como resultado dessa consulta novos documentos XML. Os documentos XML resultados podem ser sub-árvores do documento XML original ou podem apresentar estrutura diferente, inclusive com a utilização de novos elementos obtidos através de construtores de elementos. Essas características tornam a linguagem bastante

expressiva e poderosa, sendo possível, inclusive, consultar múltiplas fontes de dados XML. Embora seja possível exprimir operações de união e junção entre essas fontes de dados, a linguagem não apresenta mecanismos para resolução de conflitos. Dessa forma, as consultas XQuery assumem que as fontes de dados, embora distintas, foram concebidas dentro de um único projeto.

A XQuery permite a construção de elementos, o que torna possível contornar a falta de alguns mecanismos para resolução de conflitos. Embora os construtores de elementos forneçam alguns mecanismos para contornar o problema da heterogeneidade semântica, não existe um mapeamento formal entre atributos de fontes de dados distintas e, portanto, tais construtores não expressam de forma clara o relacionamento conceitual existente entre as fontes de dados que estão sendo integradas. Conseqüentemente, as consultas XQuery tendem a ser mais complexas que o necessário. Além disso, a linguagem XQuery não considera problemas de conexão e desconexão de fontes de dados e essa característica é fundamental em ambientes dinâmicos.

3. A Linguagem MXQuery

A linguagem MXQuery é uma extensão da linguagem XQuery, onde foram acrescentadas características que permitem manipular e integrar fontes de dados heterogêneas, tendo XML como modelo de dados comum para representação de esquemas conceituais. A linguagem XQuery é, naturalmente, orientada a consultar dados semi-estruturados e, portanto, apresenta estruturas flexíveis para identificar as informações requeridas e arranjá-las de acordo com as especificações do usuário. A linguagem MXQuery aproveitou-se dessas características e implementou declarações de mapeamento entre fontes de dados distintas para representar o relacionamento conceitual existente entre essas fontes de dados.

A idéia por trás da linguagem MXQuery é utilizar declarações de mapeamento entre os elementos originais das fontes de dados e um ou mais elementos de um documento XML que conterá o resultado da consulta. O processo de mapeamento está inserido na própria consulta, tornando, dessa forma, a MXQuery mais flexível que as propostas apresentadas em [14] e [18]. Além disso, é possível utilizar variáveis de resultado (associadas aos elementos do documento XML resultado) e variáveis das fontes de dados (associadas aos elementos das fontes de dados) em especificações de condições na cláusula "*WHERE*". Isso permite que sejam inseridos filtros (condições) sobre dados integrados e sobre dados das fontes de dados locais.

Por esse motivo, as condições existentes na consulta são processadas em dois momentos distintos. As condições sobre dados das fontes de dados locais são inseridas nas sub-consultas encaminhadas às fontes de dados, limitando, portanto, o resultado da sub-consulta sobre essas fontes de dados. As condições sobre os dados integrados são processadas após a etapa de construção do resultado global, ou seja, elementos inicialmente inseridos no resultado podem ser excluídos ou modificados de acordo com as condições sobre os dados integrados.

Por exemplo, a expressão ***WHERE $p/livro/ano > "1999"***, onde "$p" representa um documento XML resultado, só será avaliada após a integração dos dados. Por isso, poderá ser gerado um tráfego desnecessário pela rede. Elementos "livro" que possuam sub-elementos "ano" menores que 1999 são enviados para o gerenciador de consultas e, só após a etapa de integração, esses elementos são descartados. Contudo, durante a etapa de simplificação, a arquitetura MDBS tenta transformar as condições baseadas em dados integrados em condições baseadas nas fontes de dados locais, pois essa simplificação reduz a quantidade de dados trafegada pela rede. Por exemplo, a condição acima deve ser reescrita pelo gerenciador de consultas, como ***WHERE $d1/livro/ano > "1999" AND $d3/book/year > "1999"***, onde "$d1" e "$d2" identificam as fontes de dados locais (ver consulta 2 – seção 3.2.).

Diferentemente das propostas [14], [6] e [18], a linguagem MXQuery não requer a definição de contextos de uma consulta. Por esse motivo, as consultas expressas através da linguagem MXQuery são mais dinâmicas e, conseqüentemente, sofrem menor interferência da evolução dos esquemas locais, conexão e desconexão de novas fontes de dados. Isso significa que, no momento em que uma consulta MXQuery é proposta, toda a informação sobre como integrar as fontes de dados consultadas é expressa na própria consulta. Dessa forma, a evolução dos esquemas locais não compromete as definições do ambiente de integração, apenas exige novas formas de expressar as consultas MXQuery.

Uma grande dificuldade em processar consultas, sobre múltiplas fontes de dados heterogêneas que merece destaque, é o tratamento da disponibilidade das fontes de dados a serem consultadas [20, 1]. A estratégia adotada pela linguagem MXQuery para resolver esse problema é especificar as diversas fontes de dados de uma consulta *C* e, em seguida, classificá-las de acordo com sua relevância para *C*. Se uma fonte de dados é imprescindível para a consulta *C*, então ela é classificada como obrigatória, caso contrário, ela é classificada como opcional. As fontes de dados opcionais, não disponíveis durante a execução de uma consulta, não interrompem o processamento dessa consulta. Com isso, fornece-se ao processador de consultas do MDBS a capacidade de alterar a consulta original para retirar qualquer referência a essa fonte de dados opcional. Em contrapartida, as fontes de dados classificadas como obrigatórias não podem estar indisponíveis no momento da consulta, caso isso ocorra, a consulta não é executada.

[42]	EFLWORExpr	::=	EachClause (ForClause | LetClause) * WhereClause? OrderByClause? “return” ExprSingle
[144]	EachClause	::=	“each” “$” VarName “full”? (“,” “$” VarName “full”?)* DefClause
[145]	DefClause	::=	AliasClause (AliasClause)* (EqualClause)*
[146]	AliasClause	::=	“alias” PathExpr “$” VarName “null”? (“,” “$” VarName “null”?)*
[147]	EqualClause	::=	“equal” RelativePathExpr* “is” RelativePathExpr “key”? “hide”?
[43]	ForClause	::=	“for” “$” VarName TypeDeclaration? PositionalVar? “in” ExprSingle (“,” “$” VarName TypeDeclaration? PositionalVar? “in” ExprSingle)*
[45]	LetClause	::=	“let” “$” VarName TypeDeclaration? “:=” ExprSingle (“,” “$” VarName TypeDeclaration? “:=” ExprSingle)*
[122]	TypeDeclaration	::=	“as” SequenceType
[44]	PositionalVar	::=	“at” “$” VarName
[46]	WhereCaluse	::=	“where” Expr
[47]	OrderByCaluse	::=	(“order” “by” | “stable” “order” “by”) OrderSpecList
[48]	OrderSpecList	::=	OrderSpec (“,” OrderSpec)*
[49]	OrderSpec	::=	ExprSingle OrderModifier
[50]	OrderModifier	::=	(“ascending” | “descending”)? ((“empty” “greatest”) | (“empty” “least”))? (“collation” StringLiteral)?

Tabela 1. Sintaxe das Expressões EFLWOR.

3.1. Expressões EFLWOR

A idéia básica da MXQuery é estender a estrutura FLWOR presente na linguagem XQuery. Na MXQuery poderão existir expressões do tipo **EFLWOR** (*Each* – FLWOR), além de expressões FLWOR. Uma consulta baseada em uma expressão EFLWOR percorre todos os elementos das árvores referentes aos documentos XML que representam as fontes de dados participantes da consulta. O resultado da consulta é uma árvore, que é denominada árvore resultado. A árvore resultado é construída através da união, junção e/ou fusão de elementos pertencentes a documentos (fontes de dados) distintos.

Por se tratar de uma extensão da linguagem XQuery, parte da gramática da linguagem MXQuery já está descrita em [2]. A tabela 1 descreve a sintaxe das expressões EFLWOR onde a notação EBNF foi utilizada. A primeira coluna da tabela na gramática original apresenta um número seqüencial para todos as cláusulas definidas pela gramática. A tabela 1 inseriu elementos numerados de 144 a 147. O elemento 42 deve substituir o elemento da gramática XQuery original e os outros elementos presentes na tabela 1 foram repetidos por uma questão de clareza.

As expressões EFLWOR introduzidas pala linguagem MXQuery originaram cláusulas não definidas para a linguagem XQuery. Tais cláusulas são as seguintes: “EACH”, “ALIAS”, “EQUAL”, “FULL”, “NULL”, “HIDE” e “KEY”. Essas cláusulas são utilizadas para identificar as fontes de dados consultadas, especificar mapeamentos entre atributos locais (originados das fontes de dados) e atributos globais (representantes dos documentos resultado).

EACH. Essa cláusula associa valores a uma ou mais variáveis, denominadas variáveis de resultado. Essas variáveis são utilizadas para a construção de um resultado que poderá representar, na realidade, elementos XML gerados a partir da integração das fontes de dados participantes do mecanismo de integração. Por exemplo, a declaração ***EACH $p*** define que uma variável “$p” conterá o resultado de um processo de integração de elementos pertencentes a fontes de dados distintas. Regras para o processo de integração (por exemplo, regras para a resolução de conflitos) deverão ser definidas nas demais declarações da linguagem MXQuery. A variável declarada na cláusula “EACH” é um documento XML bem formado e pode ser referenciada por qualquer declaração da estrutura EFLWOR que aceite como parâmetro uma árvore de nós.

ALIAS. Essa cláusula especifica as árvores (ou sub-árvores) XML que serão utilizadas na consulta onde cada árvore referenciada pode representar uma fonte de dados distinta. Portanto, essa cláusula é utilizada para definir o escopo da consulta. Um documento referenciado na declaração “ALIAS” é associado a uma ou mais variáveis, denominadas variáveis de fonte de dados. De forma análoga às variáveis definidas pela cláusula “EACH”, essas variáveis podem ser utilizadas nas cláusulas que aceitem uma árvore de nós como parâmetro. Por exemplo, a declaração ***ALIAS document(“d1.xml”)/bib $d1*** define uma variável “$d1” que representa a fonte de dados especificada pelo documento “d1.xml”; a função “document” definida em [2] identifica o documento XML que

representa uma das fontes de dados participantes da consulta, além de indicar o ponto de entrada na floresta de nós do documento XML. Portanto, uma variável definida através da cláusula "ALIAS" pode representar qualquer sub-árvore da fonte de dados referenciada. A ordem de declaração de cláusulas "ALIAS" em uma consulta MXQuery especifica a ordem de prioridade entre as fontes de dados a serem acessadas. Dessa forma, a fonte de dados, especificada na primeira declaração "ALIAS", é a fonte de dados com maior ordem de prioridade e assim por diante. Em casos de conflito de dados, a ordem de prioridade será utilizada para resolver esse tipo de conflito segundo o seguinte critério: serão considerados válidos os valores de elementos pertencentes a fontes de dados com maior ordem de prioridade.

EQUAL. Essa cláusula especifica as regras de integração de elementos de documentos distintos onde grande parte dos conflitos é resolvida. A construção identifica a relação entre os elementos de documentos distintos e, em seguida, um novo elemento contendo o resultado da integração é gerado. Por exemplo, na especificação ***EQUAL $d1/livro $d3/book IS $p/livro***, um elemento (no caso, livro) é inserido na árvore resultado. A função desse novo elemento é integrar os elementos "livro" e "book" pertencentes a fontes de dados distintas. Nesse exemplo, os dois elementos originais são especificados e caracterizados, portanto, como sinônimos. Contudo essa especificação pode ser abreviada através da omissão do elemento da fonte de dados local identificada pela variável "$d1". Essa abreviação é possível porque, em uma declaração de cláusulas "ALIAS", existe um documento (no caso $d1) que possui um elemento "livro". A declaração abreviada seria ***EQUAL $d3/book IS $p/livro***. Na consulta exemplo (figura 1) pode ser observada a utilização do mecanismo de abreviação em uma cláusula "EQUAL". Normalmente, apenas os elementos referenciados em alguma declaração da cláusula "EQUAL" comporão a variável especificada na cláusula "EACH" (veja definição da cláusula "FULL").

FULL. A cláusula opcional "FULL" pode ser utilizada em associação com a cláusula "EACH". Essa cláusula especifica, que todos os elementos de todas as fontes de dados declaradas na cláusula "ALIAS" comporão a árvore resultado associada à variável de resultado especificada na cláusula "EACH", mesmo que não sejam referenciados na cláusula "EQUAL". Se nenhuma forma de restrição for especificada, esse tipo de declaração ocasiona a UNIÃO de todas as fontes de dados referenciadas. Diferentemente do modelo de dados estruturados (bancos de dados convencionais), onde a operação de união necessita que as fontes de dados sejam compatíveis, documentos XML não oferecem nenhuma restrição para operações de união.

NULL. A cláusula opcional "NULL" pode ser utilizada em conjunto com a cláusula "ALIAS". Essa cláusula especifica que a fonte de dados ou o documento referenciado não é imprescindível para a consulta. Dessa forma, caso a fonte de dados não esteja acessível ou não seja declarada corretamente, qualquer declaração referente a essa fonte de dados (ou algum elemento pertencente à mesma) será ignorada. Por exemplo, se uma consulta especificar a declaração ***ALIAS document("d3.xml")/lib $d3 NULL***, então o documento "d3.xml" não será considerado imprescindível para o resultado dessa consulta. Nesse caso, mesmo que a fonte de dados, contendo o documento, não esteja acessível, a consulta será processada. Da mesma forma, os elementos pertencentes ao documento "d3.xml" que aparecerem em outras declarações serão suprimidos. A declaração ***EQUAL $d1/livro $d3/book IS $p/livro*** é processada como se o elemento "$d3/book" não fizesse parte da expressão. A supressão do elemento "$d3/book" também ocorreria caso esse elemento fosse declarado incorretamente. Isso possibilita que alterações nos esquemas das fontes de dados não inviabilizem uma consulta. Esta propriedade é fundamental para ambientes como a Web e redes *ad hoc*.

HIDE. A cláusula opcional "HIDE" pode ser utilizada em conjunto com a cláusula "EQUAL". A cláusula indica que o resultado da relação declarada através da cláusula "EQUAL", mesmo compondo a variável especificada na cláusula "EACH", não deve aparecer na árvore resultado apresentada pela cláusula "RETURN". Por exemplo, na declaração ***EQUAL $d3/book/year $d1/livro/ano IS $p/livro/ano HIDE*** o elemento "livro/ano" (resultado da resolução de um conflito de sinônimos) pode ser referenciado em outras cláusulas da consulta, no entanto, ele não faz parte dos elementos apresentados pela cláusula "RETURN". O elemento "livro/ano" é um elemento virtual e seu ciclo de vida termina após a avaliação de todas as expressões condicionais presentes na consulta. A cláusula "HIDE" tem precedência sobre a cláusula "FULL", ou seja, um elemento declarado que utilize a cláusula "HIDE" não comporá a árvore resultado apresentada, mesmo tendo sido utilizada a cláusula "FULL".

KEY. A cláusula opcional "KEY" pode ser utilizada em conjunto com a cláusula "EQUAL", onde indica que os elementos especificados na declaração devem servir como identificador de equivalência entre objetos. Algumas consultas podem determinar que objetos que representem o mesmo conceito do mundo real devem sofrer uma fusão, onde uma única representação do objeto deve existir. Contudo, é necessário especificar quando dois objetos são idênticos. Isso é feito através da comparação entre propriedades de fontes de dados distintas que são indicadas pela cláusula "KEY". Considerando a declaração ***EQUAL $d1/livro/@isbn $d3/book/isbn IS $p/livro/isbn KEY***, os documentos "d1.xml" e "d3.xml" são processados originando um novo elemento "livro/isbn" que integra os elementos das fontes de dados originais. O conteúdo das fontes de dados sofrerá uma união para formar a árvore

resultado, no entanto, caso o valor associado a algum elemento “$d1/livro/@isbn” seja igual ao valor de algum elemento “$d3/book/isbn”, então esses dois objetos (considerados idênticos) sofrerão uma fusão.

3.2. Exemplo de uma Consulta MXQuery

A utilidade e aplicabilidade da linguagem MXQuery serão ilustradas nesta seção. A idéia é especificar uma consulta, de acordo com o seguinte cenário. Considere que algumas livrarias resolvem fundar um consórcio, onde seus títulos seriam disponibilizados na Web. Cada livraria gerencia sua própria fonte de dados. Muito provavelmente, tais fontes de dados são heterogêneas. Contudo, essa parceria entre as livrarias requer que os dados referentes aos seus acervos sejam “vistos” de forma integrada. Porém, nenhuma das livrarias deseja perder o controle sobre seus dados locais. Dessa forma, devem-se integrar fontes de dados heterogêneas, preservando a autonomia local de cada uma delas.

Suponha que a integração dos dados das diversas livrarias é realizada através da abordagem de MDBS e considere que os esquemas conceituais correspondentes a cada fonte de dados local serão representados em XML. Por uma questão de simplicidade, em apêndice A são representadas duas fontes de dados “d1.xml” e “d3.xml” com seus respectivos conteúdos. Entretanto, a arquitetura MDBS disponibiliza de fato documentos XML *Schema* que representam o mapeamento dos esquemas das fontes de dados locais em documentos representados no modelo de dados XML.

Consulta 1: Retornar títulos, autores e preços dos livros existentes nas livrarias, cujos esquemas conceituais estão representados pelos documentos “d1.xml” e “d3.xml”. Utilize o ISBN para especificar a equivalência entre os objetos de fontes de dados distintas. Considere ainda que os preços representados nos documentos em questão estão expressos em moedas distintas, onde U$ 1,00 equivale à R$ 2,98.

Uma possível expressão MXQuery é apresentada na figura 1. A execução da consulta retornará uma variável “$p” contendo elementos “livro” dos documentos d1.xml e d3.xml (especificados nas cláusulas “ALIAS”), cada um contendo os sub-elementos “titulo”, “autor” e “preco”. Na linha para recuperar a informação de preço, é necessário efetuar uma conversão de moeda (neste exemplo de Dólar para Real). Essa função de conversão resolve, portanto, um conflito de domínio existente entre as duas fontes de dados especificadas.

```
EACH $p
  ALIAS document("d1.xml")/bib $d1
  ALIAS document("d3.xml")/lib $d3
  EQUAL $d1/livro $d3/book IS $p/livro
  EQUAL $d1/livro/@isbn $d3/book/isbn IS $p/livro/isbn KEY
  EQUAL $d3/book/title IS $p/livro/titulo
  EQUAL $d3/book/author IS $p/livro/autor
  EQUAL ($d3/book/price * 2.98) IS $p/livro/preco
RETURN
  <biblioteca>
    $p
  </biblioteca>
```

Cláusula EQUAL abreviada

Fórmula de conversão entre moedas

Figura 1. Consulta 1 Expressa em MXQuery.

Observe que a cláusula “KEY” especifica a propriedade que deve ser utilizada para identificar objetos idênticos, onde objetos considerados idênticos devem sofrer uma fusão no processo de integração. Ainda nessa declaração, pode ser identificada a resolução de um conflito de esquema, visto que se pode comparar diretamente o atributo “livro/@isbn” com o sub-elemento “book/isbn”. Essa declaração insere o sub-elemento “isbn” na árvore resultado, como resultado da integração. Observe que as declarações de equivalência de título, autor e preço estão na sua forma abreviada, contudo, esse fato não causa nenhum impacto no resultado da consulta.

Consulta 2: Retornar os títulos dos livros existentes nas livrarias cujos esquemas conceituais estão representados pelos documentos XML “d1.xml” e “d3.xml”. Utilize o ISBN para suprimir duplicidades. Os livros devem ter sido publicados após 1999.

A expressão MXQuery mostrada na Figura 2 pode ser utilizada para expressar esta consulta. A consulta 2 é similar à consulta 1, porém foi acrescentada a declaração de equivalência do sub-elemento “ano” com “year”, resolvendo dessa forma um conflito de sinônimos. Observe que, diferentemente das declarações de equivalência de sub-elementos especificadas na consulta anterior, a declaração especificada na consulta 2 não determina que o sub-elemento “ano” componha o resultado dessa consulta, devido a utilização da cláusula “HIDE”. A idéia aqui é utilizar esse sub-elemento (após a resolução do conflito de sinônimo) apenas para o filtro especificado na cláusula “WHERE”.

Conforme os documentos apresentados no apêndice A, existe um conflito de dados entre dois dos elementos que comporão a árvore resultado dessa consulta. Dois elementos “livro” que apresentam o mesmo ISBN possuem o sub-elemento “ano” e “titulo” com valores distintos, onde o sub-elemento “ano” possui um dos valores igual a 2001 e o outro igual a 1999. No caso do sub-elemento “titulo”, o conteúdo de sub-elemento é apresentado em português no documento “d1.xml” e em inglês no documento “d3.xml”.

```
EACH $p
  ALIAS document("d1.xml")/bib $d1
  ALIAS document("d3.xml")/lib $d3 NULL
  EQUAL $d3/book IS $p/livro
  EQUAL $d1/livro/@isbn $d3/book/isbn IS $p/livro/@isbn KEY
  EQUAL $d3/book/title IS $p/livro/titulo
  EQUAL $d3/book/year IS $p/livro/ano HIDE
WHERE $p/livro/ano > "1999"
RETURN
  <biblioteca>
    $p
  </biblioteca>
```

Condição baseada em variável de resultado

Figura 2. Consulta 2 Expressa em MXQuery.

Esse conflito de dados é resolvido através da indicação da fonte de dados cujo valor de seus elementos deverá prevalecer. Essa indicação é feita através da ordem das declarações das fontes de dados consultadas. Na consulta 2, a fonte de dados especificada pela variável "$d1" é priorizada, significando que, em caso de conflito de dados o seu conteúdo será preservado. Devido a esse fato, o resultado da integração entre as fontes de dados "d1.xml" e "d3.xml" determina que o valor do sub-elemento "ano" deve ser igual a 2001 e o título em português deve constar no resultado. Nesse exemplo, se a ordem das fontes de dados na cláusula "ALIAS" fosse invertida, o resultado da consulta seria um elemento "livro" vazio. Observe que, na consulta 1, também ocorreram conflitos de dados (sub-elementos "titulo" e "preço"), que foram resolvidos da mesma forma.

A utilização da cláusula "NULL" na consulta determina que a indisponibilidade do documento "d3.xml" durante a execução da mesma não inviabiliza o resultado. Coincidentemente, o resultado apresentado é o mesmo, independente da disponibilidade de "d3.xml", no entanto, sua indisponibilidade ocasionaria um erro se a cláusula "NULL" não fosse especificada.

A condição utilizada na cláusula "WHERE" da segunda consulta é baseada em uma variável de resultado. Isso significa que, a princípio, para que essa condição seja avaliada, é necessário realizar a integração entre as duas fontes de dados e, em seguida, verificar a condição. Contudo, no processo de otimização de consultas, essa condição pode ser substituída por uma condição baseada em fonte de dados. A expressão equivalente seria: ***WHERE $d1/livro/ano > "1999" AND $d3/book/year > "1999"***. Pode-se notar que caso o processo de otimização execute essa substituição, então a fonte de dados "d3.xml" não enviará nenhuma resultado parcial (eliminando tráfego de rede desnecessário). Além disso, nenhum conflito de dados ocorrerá, pois apenas o documento "d1.xml" resultará um elemento cujo ISBN será igual a "352".

4. Estratégia de Integração

Esse trabalho propõe utilizar a tecnologia de MDBS como a estratégia de integração de fontes de dados heterogêneas. A Figura 3 apresenta um modelo abstrato da arquitetura utilizada para integrar dados armazenados em fontes de dados heterogêneas, distribuídas e autônomas.

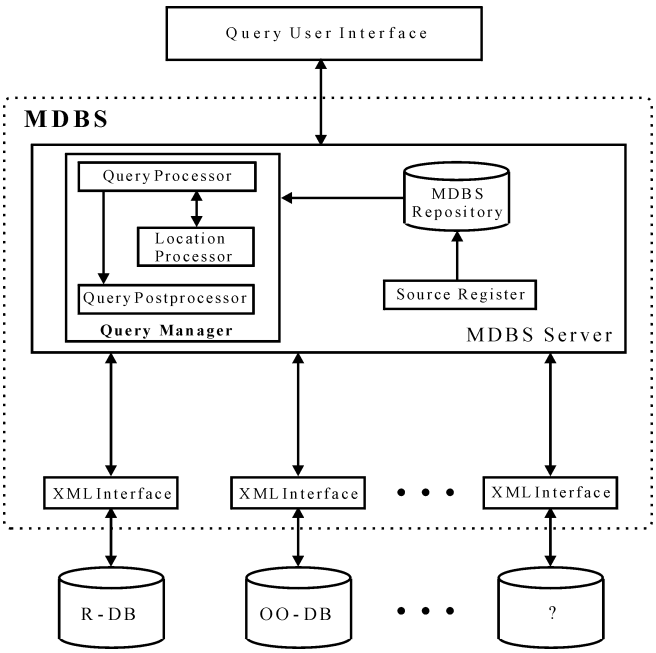

Figura 3. Arquitetura do Mecanismo de Integração.

Na Figura 3, a interface de consulta de usuário (**QUI** – *Query User Interface*) faz a intermediação entre o usuário e a arquitetura de integração, sendo de responsabilidade desse componente receber as consultas e apresentar os resultados. Um usuário, que desejar acessar múltiplas fontes de dados integradas através da arquitetura, deverá enviar ao QUI uma consulta MXQuery, denominada de consulta global. Pode-se, ainda, pensar na QUI como um agente móvel que transporta consultas MXQuery para serem executadas pela arquitetura de integração [4]. As consultas globais são enviadas, então, ao gerenciador de consultas (**QM** – *Query Manager*). A principal funcionalidade deste componente é dividir uma consulta global em sub-consultas e, em seguida, enviá-las às fontes de dados correspondentes.

O gerenciador de consulta está dividido, por sua vez, em três componentes principais, que são: processador de consulta (**QP** – *Query Processor*), processador de localização (**LP** – *Location Processor*) e pós-processador de consulta (**QPp** – *Query Postprocessor*). O processador de consulta deve receber as consultas expressas em MXQuery, identificar as fontes de dados (ou interfaces das fontes de dados) referenciadas, simplificar e otimizar

as consultas globais, gerar as sub-consultas e finalmente, encaminhar as sub-consultas para as fontes de dados correspondentes.

O processador de localização desempenha um papel auxiliar para o processador de consulta, pois ele deve localizar as fontes de dados referenciadas e indicar erros de localização. Os erros de localização podem significar a indisponibilidade de alguma fonte de dados referenciada. Essas informações são úteis no processo de simplificação de consultas, pois as fontes de dados inacessíveis devem ser excluídas da consulta. O processo de exclusão de uma fonte de dados de uma consulta deve obedecer às regras estabelecidas na própria consulta. Dessa forma, nem sempre será possível excluir a fonte de dados da consulta, nesse caso, a consulta inteira é rejeitada. Caso seja possível excluir a fonte de dados, então todas as expressões que referenciam a fonte de dados excluída devem ser excluídas. As fontes de dados que podem ser excluídas são identificadas na sintaxe do MXQuery, através da utilização da cláusula "NULL". O pós-processador de consulta será explicado mais à frente.

As sub-consultas são enviadas para as interfaces XML (*XML Interface*), onde são traduzidas para linguagem de consulta nativa e direcionadas para as fontes de dados locais correspondentes às interfaces XML. Portanto, há uma interface XML associada a cada fonte de dados local. O papel desempenhado pela interface XML será discutido posteriormente.

Depois de processada localmente, o resultado de uma sub-consulta é enviado à interface XML, onde novamente é traduzido (para XML) e encaminhado ao pós-processador de consulta. Esse componente deve utilizar as regras estabelecidas na consulta original para resolução de conflitos. Por exemplo, pode-se estabelecer uma regra para resolver problemas de sinônimo utilizando a expressão "EQUAL". Portanto, através da MXQuery, o usuário é capaz de fornecer informação para que o pós-processador de consulta resolva conflitos.

Após a resolução dos conflitos, o QPp deve combinar o resultado das diversas sub-consultas recebidas, de acordo, com as regras de construção de resultados também estabelecidas na consulta original. O resultado combinado das sub-consultas é, finalmente, encaminhado à interface de consulta de usuário.

Dois outros componentes são necessários para o funcionamento da estratégia proposta. O registrador de fonte (**SR** – *Source Register*) é encarregado de controlar a conexão e desconexão das fontes de dados participantes do MDBS. Esse componente recebe as requisições de participação no MDBS enviadas pelas fontes de dados locais, registra a localização física das fontes de dados (**URL** – *Uniform Resource Location*) e publica o esquema disponibilizado pelas fontes de dados locais.

O repositório MDBS (*MDBS Repository*) deve armazenar as informações do MDBS. Essas informações incluem os meta-dados das fontes de dados participantes, definições do MDBS, como por exemplo, as fontes de dados participantes ou consultas pré-formatadas, e localização física das fontes de dados participantes.

Uma interface XML é responsável por mapear o esquema das fontes de dados locais em um modelo de dados comum (esquema conceitual), que é apresentado na arquitetura de integração. Na arquitetura proposta, os esquemas conceituais devem ser representados através de esquemas XML, definidos através da XML *schema*, uma linguagem de definição de esquemas para dados XML [9]. Várias propostas de mapeamento de dados armazenados em bancos de dados convencionais (por exemplo, relacionais e orientados a objeto) em documentos XML têm sido publicadas e implementadas, como por exemplo, a ferramenta XDK for PL/SQL da Oracle [21]. De qualquer forma, pode-se utilizar a seguinte estratégia genérica de mapeamento: representar os objetos (ou tabelas) como elementos XML e as propriedades (ou atributos) como sub-elementos. Essa forma de mapeamento é mais transparente e garante uma maior autonomia às fontes de dados, porém o processo de integração de dados através da linguagem de consultas é sobrecarregado com as atividades de resolução de conflitos.

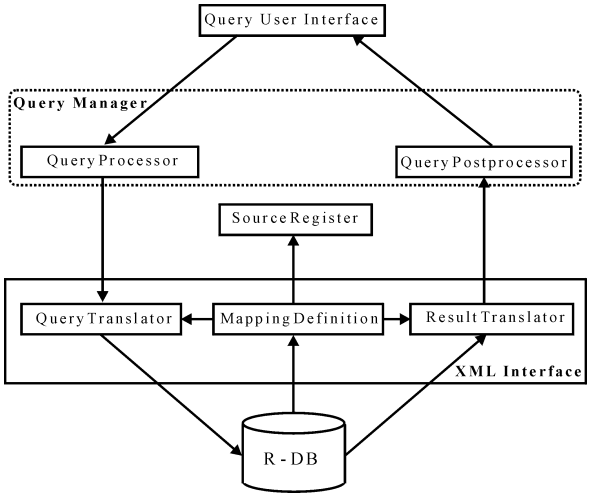

Figura 4. Detalhe da Interface XML.

A Figura 4 apresenta a interface XML com mais detalhes. A atividade de mapeamento descrita anteriormente é executada pelo componente de definição de mapeamento (*Mapping Definition*).

Além da atividade de mapeamento a interface XML deve traduzir consultas expressas no modelo de dados XML para o modelo de dados da fonte de dados local. Essa tarefa é realizada pelo tradutor de consultas (*Query Translator*). Este componente utiliza esquemas XML (definidos em XML *schema*) para efetuar o mapeamento.

O último componente da interface XML é o tradutor de resultados (*Result Translator*), que executa o papel inverso do tradutor de consultas. Esse componente deve apresentar os resultados expressos de acordo com o modelo de dados da fonte de dados local em um formato de documento XML. Por exemplo, um banco de dados relacional retorna um conjunto de tuplas para uma determinada consulta, em seguida, o tradutor de resultados deve criar elementos e sub-elementos XML que identifiquem a tabela consultada, as diversas tuplas obtidas e o conteúdo dos atributos de cada tupla. Portanto, a saída deste componente é um documento XML bem-formado.

5. Processador de Consultas

O primeiro passo executado pelo QP, ao receber uma consulta MXQuery, é checar a sintaxe e semântica da consulta recebida, onde uma árvore de expressões que simboliza a consulta original é gerada. Observando a figura 5, podem-se identificar dois componentes responsáveis por executar este primeiro passo. O primeiro faz a verificação sintática da consulta, onde uma árvore de análise obtida a partir da consulta original é comparada à gramática da linguagem MXQuery. Caso a consulta seja sintaticamente válida, o segundo componente verifica a existência dos objetos apontados na consulta no catálogo de metadados publicados. Finalmente, se todos os objetos existirem e forem declarados corretamente, é gerada a árvore de expressões de consulta global.

O otimizador global simplifica, para melhorar a performance, a consulta original avaliando diferentes planos lógicos de consulta equivalentes, que são obtidos através da aplicação de regras de transformação especificadas na álgebra que define a linguagem de consultas.

Em um banco de dados convencional, o processador de consultas deve determinar o plano físico a ser executado a partir do plano lógico selecionado durante a etapa de otimização de consultas, porém, ao integrar bancos de dados múltiplos, só se conhece a disponibilidade dos resultados parciais no momento da integração. Essa informação pode alterar a seleção do plano físico [15]. Além disso, o MDBS não possui todas as informações estatísticas necessárias a respeito das fontes de dados, conseqüentemente, é difícil optar pelo melhor plano físico antes de obter os resultados parciais.

O gerador de sub-consultas utiliza a árvore de expressões simplificada para produzir várias sub-consultas expressas em XQuery. Será produzida pelo menos uma sub-consulta XQuery para cada fonte de dados que conste na relação de fontes de dados consultadas. Como as próprias expressões XQuery podem ser simplificadas utilizando a especificação formal apresentada em [7], existe um fluxo nos 2 (dois) sentidos entre o otimizador global e o gerador de sub-consultas.

Assim, a interação entre esses dois processos determina a geração de 5 (cinco) produtos:

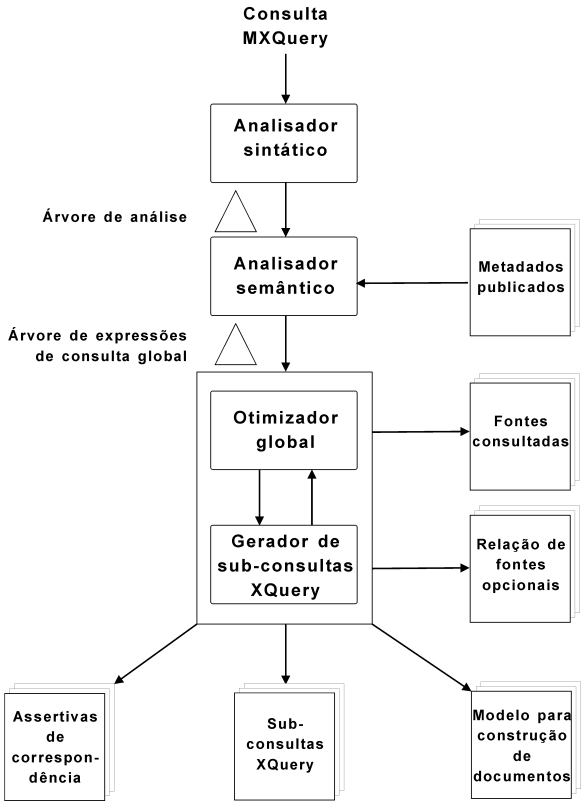

Figura 5. Produtos Gerados pelo QP.

1. Fontes consultadas. Uma lista de todas as fontes que foram referenciadas pela consulta global. Essa informação será utilizada pelo processador de localização e pelos processos de envio de consulta e recebimento de resultados;
2. Relação de fontes opcionais. A linguagem MXQuery permite a classificação das fontes de dados em obrigatórias e não obrigatórias (opcionais). Resumidamente, uma fonte de dados opcional indisponível não inviabiliza uma consulta, apenas altera parcialmente seu resultado. É produzida uma lista de todas as fontes opcionais, que será consultada quando uma fonte de dados estiver indisponível. Caso essa fonte de dados faça parte dessa relação, tal evento não interromperá a consulta;
3. Sub-consultas XQuery. É gerada pelo menos uma sub-consulta XQuery simplificada para cada fonte de dados consultada;
4. Assertivas de correspondência. É produzida uma relação com todas as assertivas de correspondência entre os elementos das fontes de dados consultadas e os elementos do documento que representará o resultado global integrado [19, 13]. As assertivas de correspondência especificam as regras para resolução de conflitos extraídas da consulta MXQuery;
5. Modelo para construção de documentos. É gerado um documento que especifica as transformações/construções que devem ser realizadas sobre os elementos originais para a produção do documento resultado.

A idéia de originar vários produtos a partir de uma consulta MXQuery é permitir a geração de sub-consultas XQuery bem simples. As transformações de elementos XML e resolução de conflitos de integração só são realizadas pela arquitetura de integração. Assim, as fontes de dados locais não necessitam realizar qualquer esforço para apresentar seus resultados. Posteriormente, o QPp utilizará os produtos originados pelo QP para construir o resultado aguardado.

Antes de enviar as sub-consultas às fontes de dados correspondentes é necessário substituir a referência lógica utilizada para identificar a fonte de dados com a localização física da referida fonte. Essa tarefa é realizada pelo processador de localização que consulta o repositório MDBS, extrai a informação necessária e substitui as referências lógicas.

Finalmente, o QP envia as sub-consultas para as interfaces XML correspondentes e aguarda o sinal de recebimento. Se decorrido um intervalo de tempo $\triangle T$ e o sinal de reconhecimento não chegar, o QP reenvia a consulta. O QP continua tentando enviar a sub-consulta durante um intervalo de tempo $\triangle T'$, porém, decorrido esse intervalo de tempo e se, ainda assim, alguma interface XML não enviar o sinal de reconhecimento, o QP executa a operação de descarte ou interrompe a consulta de acordo com os seguintes critérios:

1. Descartar. Caso a referida fonte de dados conste na lista de fontes de dados opcionais, todas as suas referências são excluídas. Assim, as regras de integração e de construção de elementos que referenciam a fonte de dados indisponível, são descartadas, bem como, a lista de fontes consultadas é atualizada. Além disso, a sub-consulta é eliminada e o processo de envio é interrompido.
2. Cancelar. Caso a fonte de dados não conste na lista de fonte de dados opcionais, o QP envia uma mensagem de erro para o QPp, envia mensagens de *"abort"* para as demais interfaces XML e descarta as tabelas temporárias (produtos do processador de consultas).

6. Conclusão

Este artigo propõe uma extensão a linguagem XQuery, denominada de MXQuery, com objetivo de utilizá-la como uma linguagem consulta (MDL) em sistemas de bancos de dados múltiplos (MDBS). A linguagem proposta permite que consultas sejam realizadas sobre fontes de dados heterogêneas. A MXQuery resolve problemas de integração de dados como heterogeneidade semântica e o tratamento de informações incompletas. A estrutura da extensão é bastante flexível, pois garante a integração de um número variável de fontes de dados com diferentes graus de autonomia.

Foi proposta ainda uma arquitetura para o processamento de consultas em sistemas de banco de dados múltiplos. A idéia é que a linguagem MXQuery seja incorporada a esta arquitetura como linguagem de banco de dados múltiplos (MDL). Atualmente, tem-se trabalhado em um protótipo da arquitetura de integração proposta neste artigo. Devido à natureza dos documentos produzidos pela arquitetura e da necessidade de manter uma independência de plataforma, tem-se utilizado a linguagem Java para implementar o processador de consultas MXQuery. Além disso, tem-se estudado a viabilidade de utilizar um banco de dados XML nativo para implementar a função do repositório MDBS. Além disso, pretende-se formalizar uma álgebra para a linguagem MXQuery a partir da álgebra proposta para XQuery. Com esta álgebra, pretende-se aproveitar um maior número de oportunidades de otimização no processamento de consultas na arquitetura proposta.

REFERÊNCIAS

[1] Abiteboul, S., Segoufin, L., Vianu, V.. Representing and Querying XML with Incomplete Information. *In Proc. of ACM PODS'01*. 2001.

[2] Boag, S., Chamberlin, D., Fernadez, M., Florescu, D., Robie, J., Siméon, J., Stefanescu, M.. XQuery 1.0: An XML Query Language. *W3C - Work in progress*. http://www.w3.org/TR/xquery/. 2003. Consultado em 12/11/2003.

[3] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E.. Extensible Markup Language (XML) 1.0 (Second Edition). *W3C Recommendation*. http://www.w3.org/TR/REC-xml. 2000. Consultado em 19/09/2001.

[4] Brayner, A., Aguiar, J.. Sharing Mobile Databases in Dynamically Configurable Environments. *In Proc. of the 15th International Conference on Advanced Information Systems Engineering*. 2003.

[5] Chamberlin, D., Robie, J., Florescu, D.. Quilt: An XML Query Language for Heterogeneous Data Sources. *International Workshop on the Web and Databases (WebDB'2000)*. 2000.

[6] Domenig, R., Dittrich, K.. A Query Based Approach for Integrating Heterogeneous Data Sources. *In Proc of. 9th International Conference on Information Knowledge Management*. 2000.

[7] Draper, D., Fankhauser, P., Fernández, M., Malhotra, A., Rose, K., Rys, M., Siméon, J., Wadler, P.. XQuery 1.0 and XPath 2.0 Formal Semantics. http://www.w3.org/TR/xquery-semantics/. 2003. *W3C Working Draft*. Consultado em 20/02/2004.

[8] Elmagarmid, A., Bouguettaya, A., Benatallah, B.. Interconnecting Heterogeneous Information Systems. *Kluwer Academic Publishers*. 1998.

[9] Fallside, D.. XML Schema Part 0: Primer. *W3C Recommendation*. http://www.w3.org/TR/xmlschema-0/. 2001. Consultado em 21/09/2001.

[10] Grant, J., Litwin, W., Roussopoulos, N., Sellis, T.. Query Languages for Relational Multidatabases. *In Proc. of 19th VLDB Conference*. 1993.

[11] Krishnamurthy, R., Litwin, W., Kent, W.. Languages Features for Interoperability of Databases with Schematic Discrepancies. *ACM SIGMOD Record, 20 (2): 40-49*. 1991.

[12] Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B., Vigier, Ph.. MSQL: A Multidatabase Language. *Information Sciences, (49)*. 1989.
[13] Lóscio, B.. Atualização de Múltiplas Bases de Dados através de Mediadores. *Tese de Mestrado, Universidade de Federal do Ceará, Brasil*. 1998.
[14] Missier, P., Rusimkiewicz, M.. Extending a Multidatabase Manipulation Language to Resolve Schema and Data Conflicts. *In Proc. of the 6th IFIP TC-2 Working Conference on Data Semantics*. 1995.
[15] Ozcan, F., Nural, S., Koksal, P., Evrendilek, C.. Dynamic Query Optimization in Multidatabases. *Bulletin of the IEEE Computer Society Techinical Committee on Data Engineering*. 1997.
[16] Özsu, M., Valduriez, P.. Principles of Distributed Database Systems. *Prentice–Hall, 2nd edition*. 1999.
[17] Robie, J., Chamberlin, D., Florescu, D.. Quilt: An XML Query Language. *Presented at XML Europe*. 2000.
[18] Sattler, K., Conrad, S., Saake, G.. Adding Conflict Resolution Features to a Query Language for Database Federations. *In Proc. of 3rd International Workshop on Engineering Federated Information Systems*. 2000.
[19] Spaccapietra, S., Parent, C., Dupont, Y.. Model Independent Assertions for Integration of Heterogeneous Schemas. *VLDB Journal, 1(1): 81-126*. 1992.
[20] Tomasic, A., Raschid, L., Valduriez, P.. Scaling Heterogeneous Databases and the Design of Disco. *In Proc. of the International Conference on Distributed Computing Systems*. 1996.
[21] XDK, Oracle XML Developer's Kit for PL/SQL. *Oracle Technology Network*. http://otn.oracle.com/tech/xml/xdk_plsql/indesx.html. 2003. Consultado em 22/10/2003.

APÊNDICE A – Documentos Exemplo

Documento: d1.xml

```
<bib>
   <livro isbn="393">
      <titulo>Princípios de SBDs Distribuídos</titulo>
      <autor>Özsu</autor>
      <autor>Valduriez</autor>
      <ano>1999</ano>
      <preco>89.00</preco>
   </livro>
   <livro isbn= "352">
      <titulo>Implementação Sistemas BDs </titulo>
      <autor>Garcia-Mollina</autor>
      <autor>Ullman</autor>
      <autor>Widom</autor>
      <ano>2001</ano>
      <editora>Campus</ editora >
      <preco>110.00</preco>
   </livro>
</bib>
```

Documento: d3.xml

```
<lib>
   <book>
      <isbn>053</isbn>
      <title>Fundamentals of Database Systems</title>
      <author>Elmasri</author>
      <author>Navathe</author>
      <year>1994</year>
      <press>Addison-Wesley</press>
      <price>54.00</price>
   </book>
   <book>
      <isbn>013</isbn>
      <title>A First Course in DB Systems</title>
      <author>Ullman</author>
      <author>Widom</author>
      <year>1997</year>
      <press>Prentice Hall</press>
      <price>45.00</price>
   </book>
   <book>
      <isbn>352</isbn>
      <title>Database System Implementation</title>
      <author>Garcia-Mollina</author>
      <author>Ullman</author>
      <author>Widom</author>
      <year>1999</year>
      <press>Prentice Hall</press>
      <price>50.00</price>
   </book>
</lib>
```