UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Mauri Ferrandin

INTEGRANDO BANCOS DE DADOS HETEROGÊNEOS ATRAVÉS DO PADRÃO XML

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação

Prof. Murilo Silva de Camargo, Dr.

Florianópolis, outubro 2002.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Mauri Ferrandin

INTEGRANDO BANCOS DE DADOS HETEROGÊNEOS ATRAVÉS DO PADRÃO XML

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação

Prof. Murilo Silva de Camargo, Dr.

Florianópolis, outubro 2002.

INTEGRANDO BANCOS DE DADOS HETEROGÊNEOS ATRAVÉS DO PADRÃO XML

Mauri Ferrandin

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Professor Fernando Ostuni Gauthier, Dr.

Banca Examinadora

_____

Professor Murilo Silva de Camargo, Dr. (orientador)

_____

Professor Roberto Willrich, Dr.

_____

Professor Rosvelter J. Coelho da Costa, Dr.

“Existem poucas coisas que

jamais poderão ser tiradas de um homem,

o conhecimento com certeza é uma delas.”

"A todos que acreditam que a educação é um fundamento

para construção de uma sociedade mais justa,

e buscam a luz da ciência construir um mundo

melhor procurando respostas para os

desafios do mundo moderno"

AGRADECIMENTOS

"Agradeço a Deus, a minha família, aos amigos,

a Universidade Federal de Santa Catarina,

ao Centro Universitário de Jaraguá do Sul (SC)."

SUMÁRIO

RESUMO

Com objetivo de organizar e estruturar o armazenamento das informações necessárias às organizações, SGBDs são utilizados a fim de prover o acesso de maneira ágil, eficiente e segura a estas informações pelas aplicações. Os dados armazenados em um SGBD são organizados de acordo com um esquema definido em cada organização, assim, quando estas precisam integrar/trocar informações armazenadas em seus respectivos bancos de dados, vários problemas surgem devido a heterogeneidade dos esquemas ou plataformas de hardware/software, necessitando de uma estrutura capaz de mediar tal intercâmbio. Para prover a integração de diversos bancos de dados heterogêneos, são utilizados os Sistemas Gerenciados de Bancos de Dados Distribuídos Heterogêneos, que controlam e possibilitam as aplicações acesso de maneira transparente aos dados distribuídos entre as bases heterogêneas.

Com a especificação do padrão XML, o mesmo passou a ser utilizado para intercâmbio de dados, uma vez que é capaz de agregar a seu conteúdo informações que o descrevem(metadados), possibilitando assim a representação de dados que não poderiam ser representadas através do modelo relacional utilizado pela maioria dos SGBDs. Com o padrão XML é possível então a criação de visões materializadas dos dados armazenados em um SGBD local e utilizar esta visão para os mais variados fins.

O presente trabalho apresenta uma proposta de um sistema capaz de prover o acesso - de maneira integrada e transparente para as aplicações – às informações armazenadas em bases heterogêneas e distribuídas, utilizando o padrão XML para representa-las através da criação de visões materializadas dos dados presentes em cada uma das bases a serem integradas, agrupando posteriormente as diversas visões em uma única visão XML. Tal integração traz a tona uma série de questões a serem tratadas, como a integridade dos dados que antes era controlada por cada um dos SGBDs e que agora precisa ser observada na visão integrada para garantir que os dados nela presentes possuam a mesma integridade, possibilitando assim que haja um serialização dos dados entre a visão e as bases distribuídas sem que ocorram problemas de integridade.

Palavras Chaves : Integração de Banco de Dados Distribuídos Heterogêneos, XML.

ABSTRACT

With the objective of organize and to structuralize the storage of information necessary to the organizations, databases are used in order to provide agile, efficient secure access to these information for the applications. The data stored in a database are structuralized and organized in accordance with a project defined in each organization, thus, when two or more of those needs to integrate or exchange informations stored in its respective databases, some problems appear due the heterogenity of projects or hardware/software plataform, needing a structure capable to mediate such interchange. To provide the integration between heterogeneous databases, Distributed Heterogeneous Databases Management Systems are used, they control and make possible the applications access in transparent way to the data distributed between the heterogeneous bases. With the specification of the XML standard, it passed to be used for interchange of data, a time that it's capable to add to its content informations that describe it (metadata), thus making possible the representation of data that could not be represented through the relational model actually used by the majority of the databases. With the XML standard, it's possible create materialized views of the data stored in a local Database Managment System and use this data for the most varied ends. The present work presents a proposal of a system capable to provide the access – in a way integrated and transparent for the applications - to the information stored in heterogeneous and distributed databases, using XML standard for represent the data through the creation of materialized views of the data stored in each one of the bases to be integrated, grouping later the diverse views in an integrated XML view. Such integration brings a lot of questions to be considered, like the integrity of data that before was controlled by evey database and that now this integrity needs to be observed in the integrated view to guarantee that those data will preserve the same integrity, making possible a serialisation between the view and the distributed databases without occur integrity problems.

Keywords: Distributed Heterogeneous Databases Systems Integration, XML.

LISTA DE FIGURAS E TABELAS

LIS*TAG*ENS DE CÓDIGO FONTE

Lista de Siglas

API	*Application Programming Interface*
APIX	*Aggregate Path IndeX*
BNF	*Backus-Naur Form*
CORBA	*Common Object Request Broker*
DCD	*Document Content Description*
DOM	*Document Object Model*
DTD	*Document Type Definition*
HTML	*Hypertext Markup Language*
JDBC	*Java Database Connectivity*
LORE	*Lightweight Object Repository*
LOREL	LORE *Language*
OMG	*Object Managment Group*
OQL	*Object Query Language*
QBE	*Query By Example*
SAX	*Simple Application for XML*
SBDD	Sistema de Banco de Dados Distribuído
SGBD	Sistema Gerenciador de Banco de Dados
SGBDD	Sistema Gerenciador de Banco de Dados Distribuído
SGBDH	Sistema Gerenciador de Banco de Dados Heterogêneo
SGML	*Standard Generalized Markup Language*
SMBD	Sistema de Múltiplos Bancos de Dados
SOX	*Schema for Object-Oriented XML*
SQL	*Structured Query Language*
URI	*Uniform Resource Indicator*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
XDR	XML *Data Reduced*
XML	*Extensible Markup Language*
XML-GL	XML *Graphic Language*
XML-QL	XML *Query Language*
XSL	XML *Stylesheet Language*
XSLT	XML *Stylesheet Transformation*

1.Introdução

O crescimento da Internet, em especial da *World Wide Web* trouxe grandes benefícios as organizações que a utilizam como meio de acesso as informações. Os diversos e variados sistemas de informações existentes estão gradualmente sendo integrados com servidores *Web* para transformarem consultas realizadas por usuários em resultados a serem exibidos em *browsers* de Internet.

No entanto, tanto as pessoas, como as organizações que geram informações para a rede utilizam diferentes maneiras de estruturá-las. Assim, quando necessário alguma transferência de informações entre elas, a falta de uma estrutura padronizada pode causar problemas de incompatibilidade entre o sistema transmissor e o sistema receptor.

Um Sistema Gerenciador de Banco de Dados (SGBD) é capaz de resolver os problemas de gerenciamento e acesso a grandes volumes de dados em uma única plataforma, mas muitos problemas surgem quando duas ou plataformas precisam trabalhar de maneira integrada. A maioria destes problemas são conseqüências da heterogeneidade semântica - dados duplicados entre estas plataformas representados de maneira diferente nos esquemas das bases de dados [Hull, 1997].

Um dos principais requisitos para a integração de sistemas de informações é a existência de um mecanismo que possa mediar e compatibilizar a troca de informações entre sistemas que utilizam diferentes formas de representações. As novas tecnologias associadas a linguagem *Extensible Markup Language* (XML) possibilitam o desenvolvimento de estruturas de mediação que atendem a este requisito.

Integrar diversas fontes heterogêneas de dados é um desafio que a anos vem fomentando pesquisas e surgimento de novos padrões a fim de tornar transparente o acesso a estas fontes para os usuários e desenvolvedores de aplicações.

A idéia central deste trabalho consiste na especificação e implementação de um sistema capaz de integrar dados de diversas fontes relacionais (bancos de dados heterogêneos distribuídos) através da utilização de visões materializadas dos dados, visões estas que utilizarão o padrão XML para organizar e armazenar os dados, e para as quais será também proposto um meio para definição de integridade referencial para os

dados nelas presentes, a fim de possibilitar que estes dados quando alterados na visão materializada possam ser sincronizados com as bases de origem sem causar violações de integridade. O sistema proposto não se preocupa com as questões e problemas referentes ao processo de atualização dos dados nas bases distribuídas, apenas propõe a manutenção da integridade referencial dos dados exportados para evitar problemas se os mesmos forem sincronizados para as bases de origem, as demais questões referentes ao processo de sincronização de visões materializadas e bases distribuídas estão fora do escopo deste trabalho.

1.1. Motivação

Diante do contexto atual no qual as pesquisas envolvendo bancos de dados e XML estão se desenvolvendo é clara a necessidade de trabalhos voltados para a questão da integração de dados armazenados em bases relacionais e dados armazenados na *Web*, pois segundo [Abiteboul et al., 2000], é através da convergência das soluções apontadas pelas tecnologias XML/dados semi-estruturados e documentos *Web*/banco de dados, que se acredita que uma tecnologia combinada para a *Web* irá emergir.

Já existem muitas pesquisas sendo desenvolvidas no intuito de solucionar os problemas da heterogeneidade[1] dos sistemas de armazenamento de dados e este trabalho terá também como foco a análise de soluções já propostas confrontando-as com a realidade atual.

A motivação para este trabalho é a crescente necessidade de um modelo baseado em uma camada de mediação capaz recuperar informações de bases relacionais heterogêneas[2] mantendo as regras de integridade na camada de mediação a fim de que os dados que estiverem materializados nesta camada possam ser sincronizados de volta a suas respectivas bases em situações que envolvam alterações de dados. A troca de informações dentro do modelo será realizada através de XML, e o modelo proposto conforme a Figura 22 é composto de vários módulos que serão detalhados no capítulo 4.4.

[1] Seja ela de hardware, sofware ou conceitual.
[2] SGBDs de diferentes fabricantes e/ou com heterogeneidade semântica em suas definições de dados.

1.2. Objetivos

O trabalho proposto terá os seguintes objetivos:

- Especificar um sistema de consulta de informações armazenadas em bases relacionais distribuídas heterogêneas através da criação de visões XML;
- Especificar um mecanismo capaz de manter/preservar a integridade dos dados exportados em casos de atualizações dos mesmos a fim de que estas atualizações possam ser propagadas para as bases relacionais.
- Pesquisar como as tecnologias XML e Java podem auxiliar na integração de fontes heterogêneas de dados;
- Implementar um protótipo do modelo proposto.

1.3. Metodologia

Para a realização deste trabalho, foram pesquisadas diversas bibliografias, tais como: livros, dissertações, teses, artigos, relatórios técnicos, documentos oficiais de congressos, *Workshops* e *site*s da Internet.

Para implementação de um protótipo do sistema também foi necessário um breve estudo sobre ferramentas de programação, APIs para manipulação de dados em documentos XML já existentes desenvolvidas por outros pesquisadores e/ou empresas atuantes no ramo.

1.4. Trabalhos Correlacionados

Existem diversos estudos e propostas na área de bancos de dados e dados semi-estruturados que estão correlacionados com este trabalho, dentre elas merecem um destaque as propostas de sistemas para integração de dados abordadas nos sistemas MIX, SilkRoute, Argos, Heros, Jupter, Pegasus entre outros. Muitos estudos de tendências futuras também merecem destaque tais como o apresentado por [Hull, 1997], [Manica, 2000] e [Silva, 2001].

1.5. Organização do Texto

O texto deste trabalho está organizado conforme o indicado a seguir. O capítulo 2 faz uma revisão do principais aspectos relacionados a bancos de dados distribuídos, tais como definição, arquitetura, terminologia, problemática, possibilidades para a distribuição dos dados. É descrita também, a definição de sistemas distribuídos heterogêneos, mostrando suas arquiteturas e terminologia bem como as principais formas de integração de bancos de dados individuais já existentes e autônomos, através de modelos que especificam ou não um esquema conceitual global. Detalha também, o processo de integração e tradução de esquemas, destacando van*tag*ens e desvan*tag*ens dos diferentes modelos apresentados.

O capítulo 3 apresenta o padrão XML, suas características principais, e os outros subpadrões a ele correlacionados, tais como as diversas linguagens para definição de esquemas (gramática) para documentos XML com comparativos entre as mesmas, estrutura lógica dos documentos e as principais linguagens para consulta a dados XML existentes com um comparativo entre as suas funcionalidades. E por último também aborda questões referentes a restrições de integridade em documentos XML.

O capítulo 4 trata basicamente da integração de fontes de dados heterogêneas através da utilização do padrão XML, demonstrando como se pode representar dados de bases relacionais através de XML, a utilização de visões XML em arquiteturas de integração, características, van*tag*ens e problemas que podem surgir mediante o emprego das mesmas para integrar fontes de dados, e por último são demonstrados algumas abordagens já existentes para integração de fontes de dados heterogêneas.

O capítulo 5 mostra a proposta de um novo modelo para integrar fontes de dados relacionais e heterogêneas utilizando o padrão XML, levando em conta a problemática da manutenção das regras de integridade da visão integrada dos dados de maneira a garantir que os dados possam ser sincronizados com as bases distribuídas sem enfrentar problemas com violações de integridade.

O capítulo 6 apresenta uma análise e interpretação dos resultado obtidos com este estudo e com a proposta de um modelo para integração de dados, van*tag*ens, desvan*tag*ens, problemas e dificuldades encontradas.

E por fim, no capítulo 7 estão a conclusão final sobre o trabalho e propostas de continuidade para o mesmo.

2.Bancos de Dados Distribuídos e Heterogêneos

Neste capítulo serão abordados os conceitos relacionados a Bancos de Dados Distribuídos Heterogêneos, sua características, van*tag*ens, desvan*tag*ens, bem com serão apresentados alguns modelos existentes que propiciam a integração de bases heterogêneas. Também serão abordadas algumas técnicas e ferramentas utilizadas para integra-los.

2.1. Conceitos básicos

Um sistema gerenciador de banco de dados distribuído (SGBDD) é um software que gerencia um banco de dados distribuído de tal modo que os aspectos da distribuição ficam transparentes para o usuário. O usuário de um sistema de banco de dados distribuído é incapaz de saber a origem das informações, tendo a impressão de estar acessando um único banco de dados.

Um sistema de banco de dados distribuído(SBDD) é como uma coleção de dados inter-relacionados que se encontram fisicamente distribuídos pelos nós de uma rede de computadores.

A Figura 1 mostra como ocorre a distribuição dos dados através de uma rede de computadores.

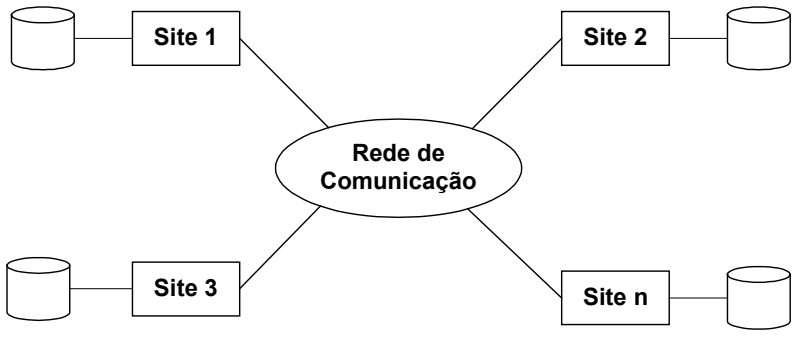

Figura 1 :Arquitetura de um banco de dados distribuído.

Cada nó de um banco de dados distribuído é capaz de processar transações locais, as quais acessam apenas dados daquele único nó, ou pode participar na execução de

transações globais, que fazem acesso a dados em diversos nós [Manica, 2001]. A execução de transações globais requer comunicações entre os nós o que implica na existência de uma infra-estrutura de rede para prover tal comunicação.

Se o projeto de um sistema distribuído é executado *top-down*, isto é, sem um sistema já existente, é conveniente desenvolver um sistema homogêneo. Todavia, em alguns casos onde a criação do banco de dados distribuído for feita pela integração de vários bancos de dados já existentes (chamamos *bottom-up*), será necessário um SGBDD heterogêneo, capaz de fornecer interoperabilidade entre os bancos de dados locais.

Existem diversas razões para construir um banco de dados distribuído, como o partilhamento de dados, confiabilidade, disponibilidade e aceleração de processamento de consultas. Entretanto, juntamente com essas van*tag*ens há diversas desvan*tag*ens, como o custo de desenvolvimento de software, maior potencial para existência de erros e crescente sobrecarga de processamento.

A principal van*tag*em de sistemas de bancos de dados distribuídos é a capacidade de dividir e fazer acesso a dados de uma maneira confiável e eficiente. Pois, se uma série de nós diferentes estão conectados, então um usuário em um nó pode ser capaz de fazer acesso a dados disponíveis em um outro nó. Cada nó é capaz de reter um grau de controle sobre dados armazenados localmente.

Em caso de uma falha em um dos nós do sistema distribuído, os nós remanescentes podem ser capazes de continuar operando, aumentando a confiabilidade e disponibilidade. Além disso, quando uma consulta envolve dados em diversos nós, é possível dividi-la em subconsultas que podem ser executadas em paralelo, acelerando seu processamento.

2.2. Requisitos funcionais de um SGBDD

Em 1987, C. J. Date, um dos primeiros projetistas de bancos de dados relacionais (junto com o Dr. E. F. Codd, autor da teoria relacional), propôs 12 regras que um SGBDD completo deveria seguir. Essas regras não representam requisitos absolutos, foram propostas somente para esclarecer as discussões sobre sistemas de bancos de dados distribuídos. No entanto, elas se tornaram largamente aceitas como um conjunto

de definições de trabalho e como critérios de um banco de dados distribuído. As 12 regras de Date são:

1. Autonomia local : cada nó participante de um sistema distribuído deve ser independente dos outros nós;
2. Não dependência de um nó central: um sistema de banco de dados distribuído não deve depender de um nó central;
3. Operação contínua: um sistema de banco de dados distribuído nunca deve precisar ser desativado;
4. Transparência e independência de localidade: os usuários do sistema não devem saber o local (nó) onde estão localizados os dados;
5. Independência de fragmentação: as tabelas que fazem parte de um sistema de banco de dados distribuído podem estar divididas em fragmentos localizados fisicamente em diferentes nós;
6. Independência de replicação: dados podem estar replicados em vários nós da rede, de forma transparente;
7. Processamento de consultas distribuído: o desempenho de uma consulta deve ser independente do local onde a mesma é executada;
8. Gerenciamento de transações distribuídas: um SGBDD deve suportar transações atômicas. As propriedades ACID (Atomicidade, Consistência, Independência e Durabilidade) das transações devem ser suportadas;
9. Independência de hardware: um SGBDD deve poder operar e acessar dados em uma variedade de plataformas de hardware;
10. Independência de sistema operacional: um SGBDD deve poder executar em sistemas operacionais diferentes;
11. Independência de rede: um SGBDD deve ser projetado para executar independente do protocolo de comunicação;
12. Independência de SGBD: um SGBDD ideal deve possuir capacidades para se comunicar com outros sistemas de banco de dados executando em nós

diferentes, mesmo se estes sistemas de bancos de dados são diferentes (heterogêneos).

2.3. Fatores negativos na utilização de SGBDD

A complexidade em sistemas distribuídos aumenta devido a vários fatores. Um deles, refere-se à distribuição dos dados. Não é essencial que cada *site* da rede possua o banco de dados completo e sim que um banco de dados resida em mais de um *site*. Portanto, é necessário definir como será a distribuição e replicação (ou não) dos dados.

A maior desvan*tag*em do sistema de banco de dados distribuído é a complexidade adicional requerida para assegurar a própria coordenação entre os nós. Devido a esta complexidade são requeridos hardware e software adicionais, o que leva a um aumento de custos de desenvolvimento, potencialidade de defeitos e da sobrecarga de processamento.

A Tabela 1 resume os principais fatores complicadores na utilização de um SGBDD.

Projeto	• como distribuir o banco de dados • distribuição dos dados replicados
Processamento de consultas	• conversão de transações de usuários em instruções de dados • problema de otimização
Controle de concorrência	• sincronização de acessos concorrentes • consistência e isolamento de efeitos de transações • gerenciamento de *deadlocks*
Confiabilidade	• como manter o sistema imune à falhas • atomicidade e durabilidade

Tabela 1 : SGBDD: fatores complicadores.

2.4. Arquiteturas para SGBDD

Uma arquitetura define a estrutura de um sistema: identificação, definição da função, e o inter-relacionamento e interações entre os componentes do sistema.

A arquitetura data-lógica é formada pelo esquema interno local de cada *site*, o esquema conceitual local de cada *site*, o esquema conceitual global e esquemas externos [Manica, 2001].

A Figura 2 representa a arquitetura data-lógica de SGBD Distribuído.

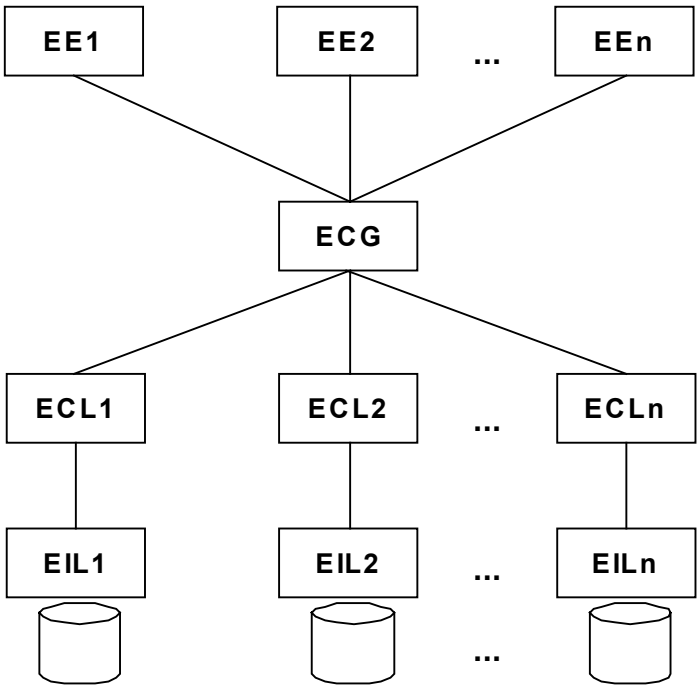

Figura 2 : Arquitetura data-lógica de SGBD distribuído.

O esquema interno local (EIL) refere-se aos aspectos relacionados ao armazenamento físico dos dados do *site*. O esquema conceitual local (ECL) refere-se à estrutura lógica (informações) do banco de dados.

O esquema conceitual global (ECG) é formado pela união dos esquemas conceituais locais, permitindo uma visão global dos dados. Finalmente o nível mais externo, os esquemas externos (EE) são as visões definidas aos usuários globais.

Esta arquitetura é denominada ponto-a-ponto (*peer-to-peer*) devido ao fato de que cada *site* possui o SGBD completo, diferente da arquitetura cliente servidor que concentra o gerenciamento dos dados em servidores, enquanto nos clientes ficam as interfaces e aplicações.

Quando o projeto do banco de dados distribuído é realizado a partir de base de dados já existentes o chamamos de *bottom-up*. Deste modo, surge uma nova arquitetura data-lógica de multi-SGBD.

A Figura 3 mostra a Arquitetura data-lógica de multi-SGBD. A maior diferença entre esta arquitetura e a data-lógica é forma do mapeamento entre esquemas.

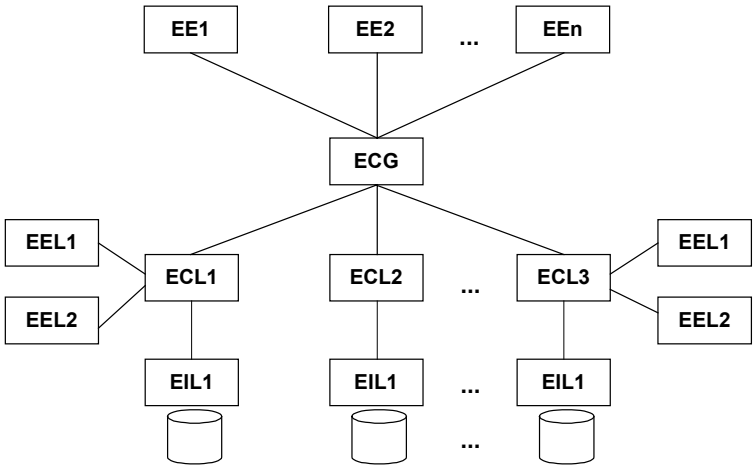

Figura 3 : Arquitetura data-lógica de multi-SGBD

2.5. Sistemas Homogêneos e Sistemas Heterogêneos

Sistemas de bancos de dados homogêneos são aqueles que possuem o mesmo software gerenciador de banco de dados em todos os *sites* integrantes deste sistema na rede.

Diversas aplicações de banco de dados têm sido desenvolvidas requerendo dados de uma variedade de sistemas de bancos de dados preexistentes, localizados em vários ambientes heterogêneos de hardware e software.

A manipulação de informações localizadas em bancos de dados heterogêneos requer uma camada adicional de software no topo dos sistemas de bancos de dados existentes. Essa camada de software é chamada de Sistema Gerenciador de Bancos de Dados Heterogêneos (SGBDH).

Considera-se um SGBDD heterogêneo [Özsu e Valduriez, 1999] aquele que usa pelo menos dois tipos de SGBDs diferentes. Portanto, um SGBDH fornece transparência não só da distribuição dos dados, mas também dos diferentes sistemas que o usuário acessa.

Um SGBDH fornece uma visão integrada que esconde diferenças de estruturas e distribuição do vários bancos de dados. Esta visão integrada é apresentada como uma visão global do banco de dados (esquema conceitual global) e é expressa em algum modelo de dados comum aos SGBDs locais, como o orientado a objetos, entidade-relacionamento ou o modelo relacional.

O SGBDH é responsável pelo mapeamento de dados e operações entre o banco de dados virtual (esquema conceitual global) e o banco de dados local (esquema conceitual local), por resolver diferenças entre modelos, esquemas e sistemas, e por gerenciar as transações distribuídas e o controle de concorrência [Özsu e Valduriez, 1999].

2.6. Banco de Dados Distribuído Heterogêneo

Em sistemas distribuídos heterogêneos, as nomenclaturas mais comumente utilizadas são: sistemas *multidatabase*, sistemas federados e sistemas legados. O termo sistema gerenciador de banco de dados distribuído heterogêneo é uma generalização destas arquiteturas.

2.6.1. Sistemas Multidatabase

Um sistema com múltiplos bancos de dados: *multidatabase* (SMBD) é um tipo especial de sistema de banco de dados distribuído. É formado por uma coleção coerente e integrada de dados que logicamente aparenta ser um único banco de dados mas é implementado fisicamente em vários bancos de dados.

Cada banco de dados participante de um SMBD é autônomo. Os usuários locais dos bancos de dados participantes continuam usando as suas aplicações locais no banco de dados sem nenhuma alteração pela sua participação no SMBD.

Os bancos de dados que participam no SMBD são geralmente heterogêneos e os usuários não precisam saber como ou de onde os dados são acessados. Em um SMBD existem usuários locais e globais. A Figura 4 detalha a arquitetura de um SMBD.

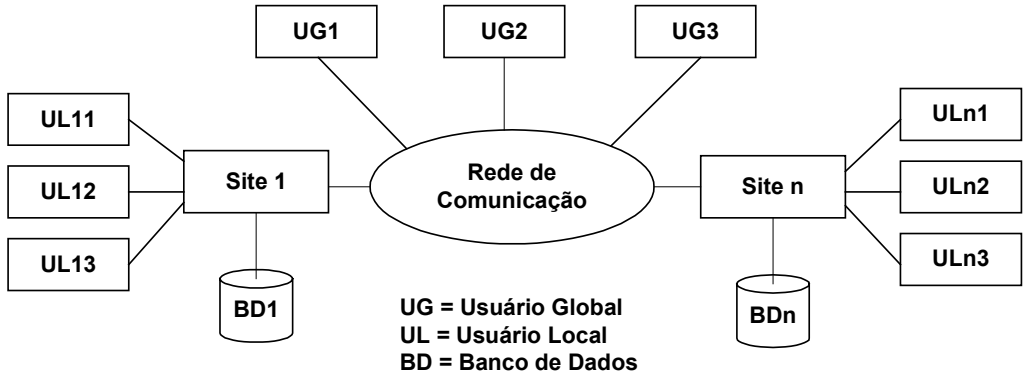

Figura 4 : Sistema Multidatabase (com usuários locais e globais)

2.6.2. Sistema Federado e Não Federado

Consideram-se os sistemas de bancos de dados federados [Sheth e Larson, 1990] como um subcaso de sistemas *multidatabase*, sendo que os sistemas federados podem ser classificados em :

- sistemas fracamente acoplados : são aqueles que não possuem um esquema global dos dados;
- sistemas com acoplamento forte : são compostos por conjuntos de SGBDs componentes, heterogêneos, cooperativos mas autônomos, integrados de tal forma na federação que consultas e atualizações podem ser realizadas de forma transparente à localização dos dados e aos caminhos de acesso. Isto é viabilizado pela presença de um esquema global.

Os sistemas não federados são *multidatabase* que não possuem usuários locais. O esquema conceitual global é definido através da união de todos os esquemas locais. Desta forma, todos os dados dos bancos de dados locais são compartilhados. Em um sistema *multidatabase* federado, os bancos de dados locais são semi-autônomos, pois operam independentemente e participam da federação compartilhando parte de seus dados.

O modelo de banco de dados federado é mais flexível pois suporta a autonomia dos bancos de dados participantes da federação. Para organizações descentralizadas, este modelo é ideal porque cada componente do banco de dados controla o acesso a seus dados.

2.6.3. Sistemas Legados

Sistemas legados [Silva, 1994] (*Legacy Systems*) são aqueles sistemas que estão em uso por muito tempo, que atendem aos requisitos dos usuários e são de difícil substituição ou porque a reimplementação de seu código é inviável financeiramente ou porque eles são imprescindíveis, já que esses sistemas não podem ficar sem execução por muito tempo.

Na maioria das vezes, os sistemas legados foram desenvolvidos em linguagens procedurais, não implementam abstração de dados e não possuem documentação, exceto

o código fonte. Tudo isso dificulta a adição de novas funcionalidades e a realização de manutenção no sistema.

Em [Brodie e Stonebraker, 1995] define-se Sistemas Legados como aqueles sistemas que contêm dados valiosos, mas que carecem de poder ou agilidade para satisfazer as necessidades atuais da organização. A necessidade de sobrevivência dos sistemas legados, em sua grande maioria, faz com que várias alternativas tenham sido propostas para resolver, ou pelo menos minimizar esse problema.

2.7. Técnicas e Ferramentas de Integração

Normalmente, os dados que empresas e instituições públicas de médio e grande porte desejam compartilhar são dados que estão em bancos de dados heterogêneos já existentes, o que torna a interoperabilidade ainda mais complexa.

Nestes casos, os bancos de dados individuais já existentes são autônomos e é necessário projetar uma forma ideal para integrar tais sistemas. Este projeto de integração executado a partir de base de dados existentes é chamado de *bottom-up*, e diferentes autores apresentam várias alternativas.

A integração de bancos de dados [Özsu e Valduriez, 1999] é o processo no qual informações dos bancos de dados participantes são integrados para formar um único e coeso multidatabase. Em outras palavras, é o processo de projetar um esquema conceitual global a partir dos esquemas locais de cada banco de dados participante do multidatabase.

A dificuldade na definição do modelo de dados utilizado pelo SGBD heterogêneo [Silva, 1994] é conseqüência da necessidade de se escolher um modelo de dados com poder de expressão suficiente para capturar a semântica dos dados expressa pelos esquemas locais dos SGBDs.

Existem propostas que não utilizam o esquema conceitual global para integrar múltiplas bases de dados. Há discussões se o esquema conceitual global deve existir ou não em sistemas multidatabase. Portanto, existem dois modelos que podem ser utilizados para integrar bancos de dados: arquiteturas que especificam um esquema conceitual global e arquiteturas que não especificam um esquema conceitual global.

2.7.1. Integração através de modelos que especificam um esquema conceitual global

Uma alternativa para integração de bancos de dados [Özsu e Valduriez, 1999], [Bell e Grimson, 1992] é através da especificação de um esquema conceitual global a partir dos esquemas conceituais locais. Bell e Grimson, classificam os sistemas federados que possuem um esquema global como *fortemente acoplados*. Portanto, um SBDD heterogêneo fortemente acoplado é composto por um conjunto de SGBDs componentes, integrados de forma que a localização dos dados e os caminhos de acesso são transparentes aos usuários.

A construção de um esquema global é uma tarefa difícil e complexa. O esquema global pode ser formado pela união de esquemas locais conforme a Figura 5.

Deste modo, o esquema conceitual global é um subconjunto da união de todos os esquemas conceituais locais, pois é formado apenas por parte dos esquemas conceituais locais. Em ambas alternativas as visões para usuários que requerem acesso global são definidas a partir do esquema conceitual global.

A maior diferença entre o projeto do esquema conceitual global em sistemas distribuídos e este tipo de sistema é que, no primeiro, o mapeamento ocorre do esquema conceitual local para o esquema global. No segundo, o mapeamento é ao contrário, do esquema global para o conceitual. Assim, o projeto de um sistema *multidatabase* é normalmente *bottom-up*, enquanto que nos sistemas distribuídos é *top-down*.

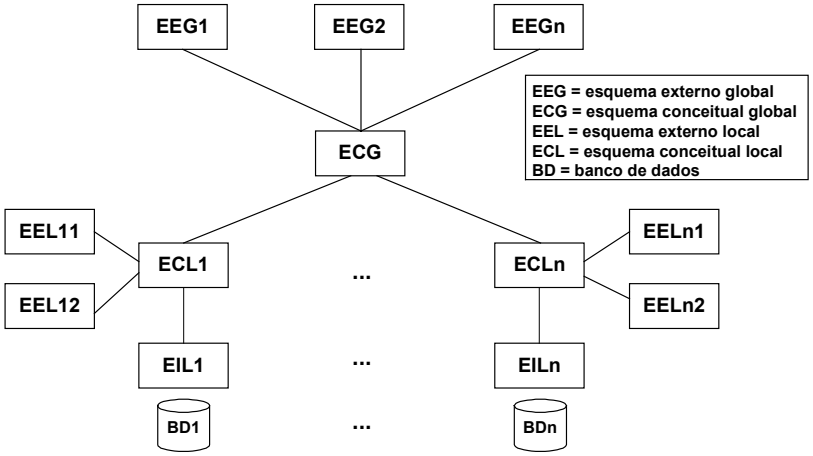

Figura 5 : Arquitetura com esquema conceitual global

2.7.2. O processo de integração e tradução de esquemas

O processo de integração ocorre em dois passos: tradução e integração de esquemas. A tradução só é necessária se os bancos de dados forem heterogêneos (cada esquema local foi definido usando modelo de dados diferentes). Nesta primeira etapa, o esquema conceitual de cada banco de dados é traduzido para um esquema intermediário padrão. O esquema intermediário corresponde ao esquema local traduzido para um modelo de dados de uso comum no SGBDH.

Na fase de integração de esquemas, realizada em seguida ao processo de tradução, ocorre a integração dos esquemas intermediários gerando um esquema conceitual global.

Integração segundo [Özsu e Valduriez, 1999], é o processo de *identificar* os componentes de um banco de dados que estão relacionados com um outro, *selecionar* a melhor representação para o esquema conceitual global, e, finalmente, *integrar* os componentes de cada esquema intermediário. A Figura 6 ilustra os processos de tradução e integração.

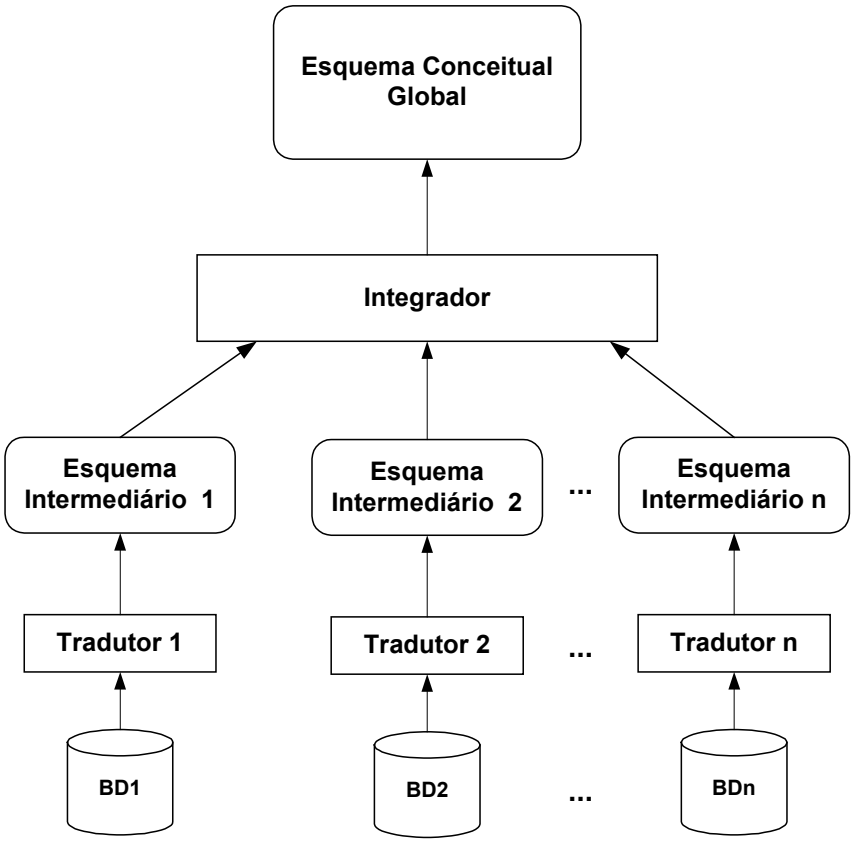

Figura 6 : Integração de Bancos de dados: tradução e integração

A integração de esquemas envolvem duas tarefas [Yan et. al., 1997]: homogeneização e integração. Na homogeneização, são tratados os problemas de heterogeneidade semântica e estrutural. Problemas semânticos referem-se ao significado, interpretação e como os dados são usados.

O problema mais importante de heterogeneidade semântica é o de conflito de nomes: *sinônimos* (duas entidades com nomes diferentes, mas com o mesmo significado) e *homônimos* (duas entidades com o mesmo nome e com significados diferentes). Existem vários métodos alternativos para lidar com conflitos de nomes.

Alguns dos problemas de heterogeneidade semântica e estrutural tratados na homogeneização não são possíveis de serem implementados. Isso faz com que seja essencial a intervenção humana na solução destes.

Após a homogeneização dos esquemas é feita a integração. Os esquemas dos múltiplos bancos de dados (agora esquemas intermediários) são combinados em um único esquema conceitual global e reestruturados da melhor forma possível.

Muitas vezes, a heterogeneidade entre os esquemas a serem integrados é muito grande, tornando inviável o investimento no desenvolvimento do esquema global, principalmente quando o número de pesquisas globais são relativamente baixas.

As principais desvan*tag*ens deste modelo são a difícil automação, necessidade de intervenção humana para a solução de conflitos semânticos e/ou estruturais, e a não adaptabilidade do sistema que faz com que toda vez que ocorram mudanças nos esquemas locais a integração deva ser refeita.

2.7.3. Mediadores

Mediador é um componente de software que explora conhecimento codificado sobre conjuntos ou subconjuntos de dados para criar informação útil para uma camada de aplicações [Wiederhold, 1992]. Uma aplicação que requer dados de fontes heterogêneas necessita de uma camada intermediária de software que faça a mediação entre ela e as fontes de dados, esta camada de mediação terá tarefas como abstração, combinação e explicação dos dados.

Os mediadores tem como principal funcionalidade a disponibilização de dados para aplicações, agindo como uma interface inteligente para lidar com problemas de representação e abstração presentes nas diferentes fontes de dados, eles apresentam regras ativas e contêm estruturas de conhecimento que guiam as transformações de dados, podendo inclusive manter resultados intermediários que podem ser consultados.

As principais funcionalidade de um mediador são:

- Transformações e geração de dados de bases de dados através da utilização de visões e *templates* de objetos reorganizando os dados de maneira apropriada para serem acessados pelas aplicações;
- Disponibilização de informações textuais como aplicação de padrões para texto visando um melhor entendimento de suas informações. Esta funcionalidade é muito importante na organização de fontes de dados semi-estruturados, como os dados presentes na *Web*;
- Armazenamento de dados derivados afim de melhorar a eficiência reduzindo o acesso as fontes e mantendo conhecimento processado para uso posterior. Para a implementar esta funcionalidade é preciso manter o controle de integridade dos dados.

A presença de mediadores pressupõe a existência de uma arquitetura básica em três camadas, conforme a Figura 7.

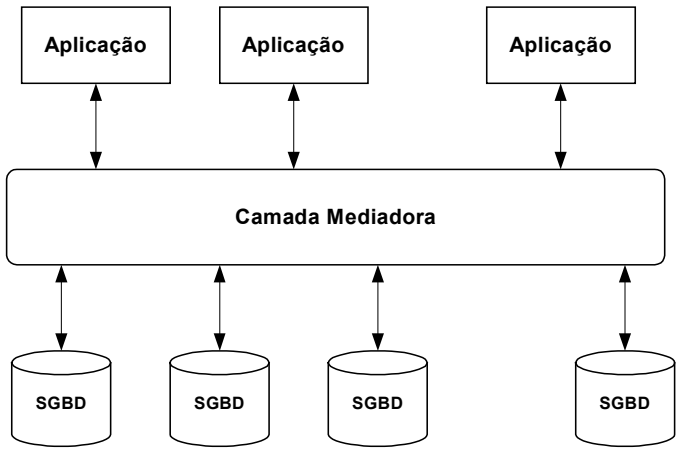

Figura 7 : Arquitetura Básica dos Mediadores.

A camada superior é composta pelas aplicações independentes, a camada intermediária é formada por múltiplos mediadores, gerenciados por especialistas em

domínios do conhecimento, sendo que o número de mediadores depende da heterogeneidade dos dados necessários a aplicação, e a camada inferior é composta de várias fontes de dados gerenciadas por administradores de dados.

A interface aplicação-mediador deve ser uma linguagem declarativa e extensível que permita a incorporação de novas funções de modo a prover a flexibilidade e a interação com novos mediadores. A interface mediador-fonte deve ser uma linguagem de acesso a dados.

Alguns aspectos devem ser levados em conta no projeto de uma arquitetura que envolva o uso de mediadores:

- A manutenção do mediador deve ser feita por um pequeno grupo de especialistas, responsáveis pela manutenção das suas regras de transformação;
- Os dados mantidos por um mediador podem servir de entrada para outros mediadores ou serem consultados por usuários ou aplicações finais;
- Mediadores podem ser especializados para prover melhor extensibilidade, facilitar a manutenção e oferecer mais opções as aplicações, assim, a camada intermediária pode ser formada por uma hierarquia de especializações de mediadores com relacionamentos de associação para troca de informações;
- Eventos (*Triggers*)[3] podem ser disparados dos sistemas gerenciadores das fontes de dados para os mediadores relacionados de modo a manter a integridade dos dados na ocorrência de alterações nos mesmos ou na sua estrutura;
- Linguagens de acesso a mediadores são objetos de pesquisas, devendo incluir capacidades funcionais da linguagem SQL, iteração, teste e ranqueamento.

2.7.4. *Wrappers*

Um *wrapper* ou tradutor, é um componente de software que converte dados e consultas (*query*) de um modelo para outro [Papakonstantinou et al., 1995]. Neste caso, uma aplicação (que pode ser um mediador), solicita ao *wrapper* consultas em uma

[3] Triggers são instruções de processamento armazenadas no próprio SGBD que são executadas automaticamente mediante condições específicas.

linguagem de consulta comum (SQL, OQL, XML-QL), e o mesmo converte esta consulta para uma linguagem de consulta suportada pelo banco de dados ao qual está ligado, depois recebe o resultado desta consulta e o converte para um formato suportado pela aplicação ou mediador.

A Figura 8 detalha uma arquitetura baseada em mediadores com a utilização *wrappers*.

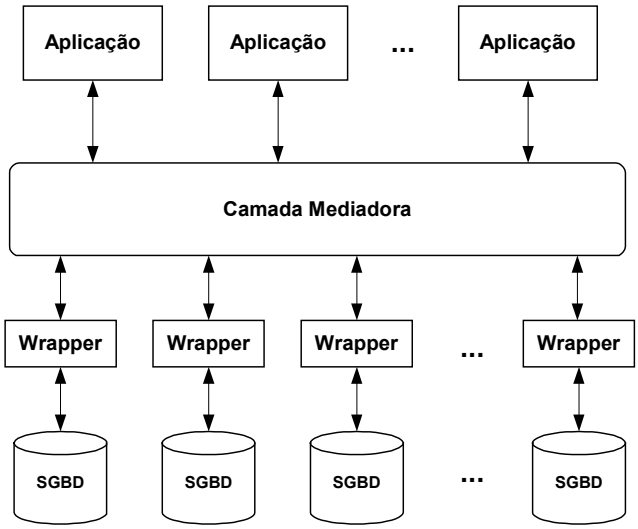

Figura 8 : Arquitetura Básica dos Mediadores com auxílio dos *wrappers*.

Podemos definir como função básica de um *wrapper* o trabalho de exportar para cada fonte de dados, informações sobre o esquema de dados e dados, além de traduzir as consultas de uma linguagem para outra.

2.8. Exemplos de Sistemas Gerenciadores de Bancos de Dados Heterogêneos

Existe um grande número de SGBDHs bem como novos projetos sendo desenvolvidos. Apesar das diferenças, todos eles possuem como principal objetivo desenvolver uma arquitetura e ferramentas que propiciem interoperabilidade entre sistemas de banco de dados heterogêneos. A Tabela 2 apresenta os principais sistemas gerenciadores com suas características e nas subseções seguintes serão detalhadas as arquiteturas dos principais SGBDHs existentes.

Sistema	Tipo de Acopla-mento	Modelo de Dados	Lingua-gem de Acesso Global	Características
Multidatabase	Forte	Orientado a objetos	Daplex	O multidatabase tem como restrição o fato de não fazer um controle ideal da consistência dos bancos de dados.
Jupter	Forte	Orientado a objetos	JIL	Os SGBDs locais são encapsulados como clientes e servidores Orbix. Solicitações remotas são tratadas por chamadas ao Jupter, que estabelece um objeto representante local para a solicitação remota.
HEROS	Forte	Orientado a objetos	OQL (da OMG)	Utiliza para o controle de concorrência global um método baseado no Método de Tíquete Implícito estendido para possibilitar a garantia de serialibilidade das transações globais.
DDTS	Fraco	Orientado a objetos	GORDAS	DDTS ainda possui vários problemas a serem solucionados, como otimização de consultas, controle da concorrência, regras de integridade, etc.
Pegasus	Fraco	Íris e Orientado a objetos	HOSQL (extensão da OSQL)	Trata conflitos e otimização de consultas.
Vodak	Fraco	Orientado a objetos	VML	Utiliza metaclasse, novas idéias para gerenciamento de transações para resolver conflitos.

Tabela 2 : Características dos SGBDH.

2.8.1. Multidatabase [Buretta, 1997]

Multidatabase é um software desenvolvido pela Computer Corporation of America para recuperação de dados de bancos de dados heterogêneos. Este sistema provê ao usuário uma visão uniforme do banco de dados e uma linguagem de manipulação de dados chamada Daplex.

O Multidatabase fornece um alto nível de interface para aplicações somente para leitura mantendo transparência da localização e heterogeneidade dos SGBDs participantes. O principal objetivo deste sistema gerenciador é prover uma interface para sistemas preexistentes sem modificar seus softwares.

A arquitetura do multidatabase demonstrada na Figura 9 é composta por dois principais componentes:

- gerenciador de dados global (GDG) : responsável pelas consultas globais;
- interface de banco de dados local (IDL) : responsável pela interface dos SGBDs nos vários *site*s.

O esquema global oferece um visão integrada do banco de dados distribuído e é acessado pela linguagem Daplex. Cada *site* possui o componente de interface local que é um intermediário entre o esquema local do *site* (acessado pela linguagem do BDs locais) e um esquema intermediário local (acessado em Daplex).

O gerenciamento de consultas é executado da seguinte maneira. Quando uma consulta global é executada, o GDG decompõe a consulta em várias subconsultas Daplex sobre os esquemas locais e o banco de dados auxiliar. O esquema auxiliar descreve dados necessários para o mapeamento entre os esquemas. Também é de responsabilidade do GDG montar os resultados parciais e retorná-los ao usuário.

O IDL (interface de banco de dados local) é responsável por aspectos relacionados especificamente aos bancos de dados locais. Ele traduz as consultas escritas nas linguagens dos BDs locais para Daplex e vice-versa.

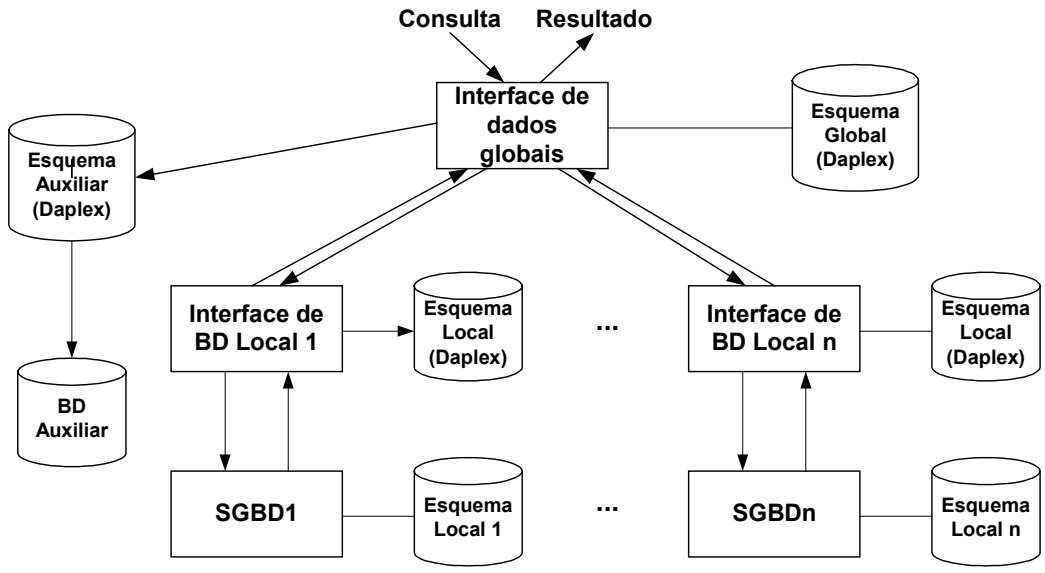

Figura 9 : Arquitetura do SGBDD-H Multidatabase

O GDG é composto por cinco módulos : transformador, otimizador global, decompositor e monitor enquanto o IDL é composto por três: interface de rede, otimizador local, e tradutor.

O transformador pega a consulta global em Daplex como entrada e produz como saída uma consulta global Daplex que referencia o esquema local. O otimizador global utiliza a saída do transformador para gerar um plano de consulta através do algoritmo de otimização SDD-1[4].

O decompositor decompõe a consulta em subconsultas e o filtro elimina de cada subconsulta aquelas operações que não são suportadas pelo SGBD correspondente. Estas operações removidas serão executadas à parte. O monitor controla a execução da consulta.

A interface de rede em cada *site* é o módulo responsável por transmitir as consultas e seus resultados. O tradutor finalmente traduz a consulta para a linguagem do banco de dados local.

O *multidatabase* tem como restrição o fato de não fazer um controle ideal da consistência dos bancos de dados. Porém, várias soluções estão sendo propostas para lidar com inconsistências. A mais utilizada é agregar funções próprias para controle da inconsistência de dados.

2.8.2. Projeto Jupter [Murphy e Grimson, 1995]

O projeto Jupter tem como objetivo básico permitir que sistemas autônomos e, possivelmente, heterogêneos cooperem e compartilhem informação de uma forma controlada.

Sua arquitetura consiste em um conjunto de serviços e uma linguagem para múltiplos bancos de dados, a JIL- *Jupter Interoperator Language* que permite que provedores de informação construam sistemas de informação fracamente acoplados, autônomos e interoperáveis.

A Figura 10 detalha os componentes da arquitetura do Jupter.

[4] SDD-1 (System for Distributed Databases), é um protótipo de um sistema de banco de dados distribuído desenvolvido pela Computer Corporation of America. No SDD-1, partições de dados distribuídos através de uma rede são replicados em múltiplos *site*s. O controle de concorrência do SDD-1 garante a consistência das bases em um ambiente de distribuição.

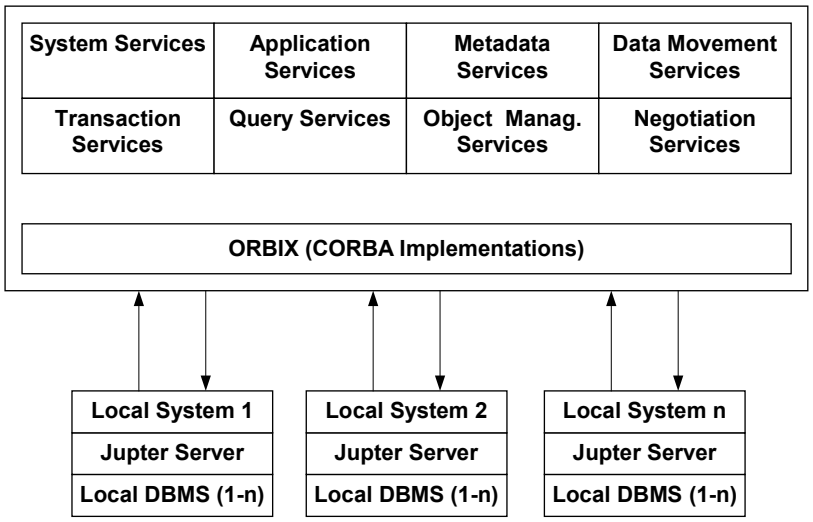

Figura 10 : Arquitetura do Jupter.

A arquitetura de esquemas possui quatro níveis: esquema local, esquema de participação, esquema de exportação e esquema federado. Os componentes da arquitetura do Jupter são: *system services, application services, dictionary (metadata), query services, transaction services, data movement services, object services e negotiation services*.

2.8.3. HEROS - HetERogeneous Object System [Castro, 1998]

HEROS - HetERogeneous Object System é um SGBDH orientado a objetos desenvolvido no departamento de informática da PUC-Rio, sob o patrocínio do CNPq. Possui acoplamento forte, que provê aos usuários transparência de localização e replicação das informações acessadas.

O modelo de dados utilizado no HEROS para expressar o esquema global é o modelo de objetos, escolhido devido a sua expressividade e minimalidade. Para representar a informação (sobre os SGBDs componentes) necessária para permitir o acesso e interoperação (meta-informação) é utilizada a hierarquia de classes. Desta forma, representa-se no esquema do HEROS, não somente os esquemas dos sistemas a serem integrados, mas também as próprias características dos modelos de dados e SGBDs dos componentes, com as regras de mapeamento entre estes e o modelo global da federação.

Deste modo, é possível a extensibilidade da federação, o que permite que qualquer novo sistema componente possa ser integrado à federação, sem a necessidade de

alterações na estrutura base do HEROS. Para integrar um novo componente, cujas características de modelo de dados ou sistema de gerência ainda não existam na federação, basta defini-lo através da criação de classes que descrevam suas características, juntamente com suas respectivas regras de mapeamento, deixando, então, que o próprio HEROS faça a tradução de esquemas automaticamente [Castro, 1998].

A arquitetura de esquemas do HEROS é dividia em quatro níveis conforme demonstrado na Figura 11: esquema local (EL), esquema de exportação (EExp), esquema global (EG) e esquema externo (EE).

O esquema local é o próprio esquema do SGBD componente, expresso no seu próprio modelo de dados. Esquema de exportação é o esquema local do SGBD componente expresso no modelo de dados do HEROS. Esquema global é obtido pela integração de todos os esquemas de exportação e esquema externo representa uma visão global do esquema integrado do HEROS.

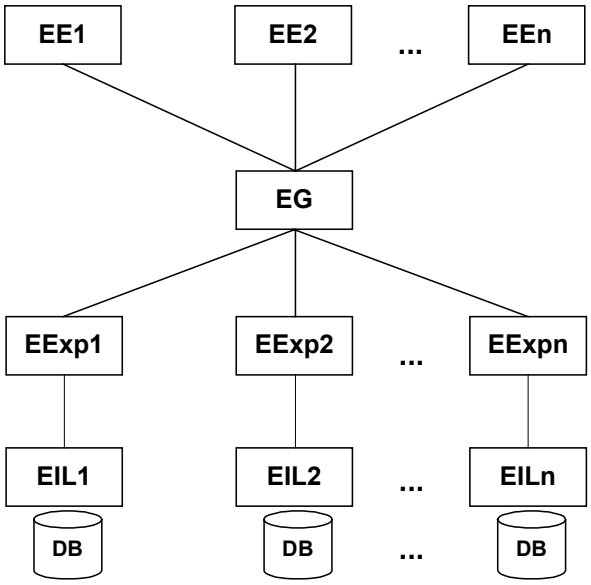

Figura 11 : Arquitetura de esquemas do HEROS

Conforme mencionado, o HEROS usa um modelo de dados orientado a objetos, o que permite que tudo seja modelado através da representação de objetos, desde a meta-informação até as instâncias dos bancos de dados. Os elementos do modelo de dados HEROS são representados na Figura 11.

O controle de concorrência em um SGBDH deve ser efetivado tanto no nível global quanto no nível local. Para o controle de concorrência local no HEROS, tendo em vista que a autonomia dos SGBDs componentes impede a interferência do gerente global no controle de transações, foram restringidos os tipos de SGBDs que podem participar, através de requisitos que devem ser atendidos pelos protocolos empregados no controle da concorrência. Cada SGBD componente deve efetuar o controle da concorrência com o uso do protocolo 2PL[5] estrito [Bernstein et al., 1987].

Para o controle de concorrência global, o HEROS apresenta um método baseado no Método de Tíquete Implícito – ITM [Georgakopoulos et al.,1994], estendido para possibilitar a garantia de serialibilidade das transações globais mesmo no caso de ocorrência de falhas acrescentando ao mecanismo original um mecanismo para controle de acessos baseado também no protocolo 2PL.

2.8.4. DDTS - Distributed Database Testbed System [Buretta, 1997]

DDTS foi desenvolvido por Honeywell Corporate Computer Science Center. O projeto enfatiza a modularização e flexibilidade e é composto por subsistemas que provêem serviços de interface com o usuário, tradução de consultas e execução distribuída. Como no *Multidatabase* (abordado na 2.8.1), ele é capaz de integrar SGBDs e prover algumas facilidades adicionais.

A arquitetura do DDTS consiste em um conjunto de processadores de aplicação (application processors: AP) e processadores de dados (data processors DP). Os Aps controlam a interface com os usuários e gerenciam aplicações enquanto os DPs gerenciam dados. Ambos são alocados a processadores físicos nos *sites* durante a configuração do sistema. Um subsistema de comunicação transfere mensagens entre Aps e DPs.

Sua arquitetura em cinco níveis utiliza um esquema global que é uma descrição relacional de toda a estrutura dos bancos de dados. A linguagem de manipulação de dados utilizada é GORDAS.

[5] 2PL (Two-Phase Locking) Algoritmo de controle de bloqueios em SGBDs baseado em duas fases.

Seus componentes conforme a Figura 12 são divididos entre Aps e DPs. Processadores de aplicação (Ap) incluem quatro módulos: interface, tradutor e controlador de integridade, planejador de acesso e monitor de execução distribuída.

Processadores de dados (DP) incluem dois módulos: o monitor de execução local e o módulo de operação local. A interface provê a interação do usuário com DDTS. Este componente fornece funções para armazenamento, edição e execução de aplicações. O tradutor traduz uma consulta na linguagem GORDAS para álgebra relacional. Informações para mapeamento são armazenadas no esquemas de representação, conforme ilustra a Figura 12.

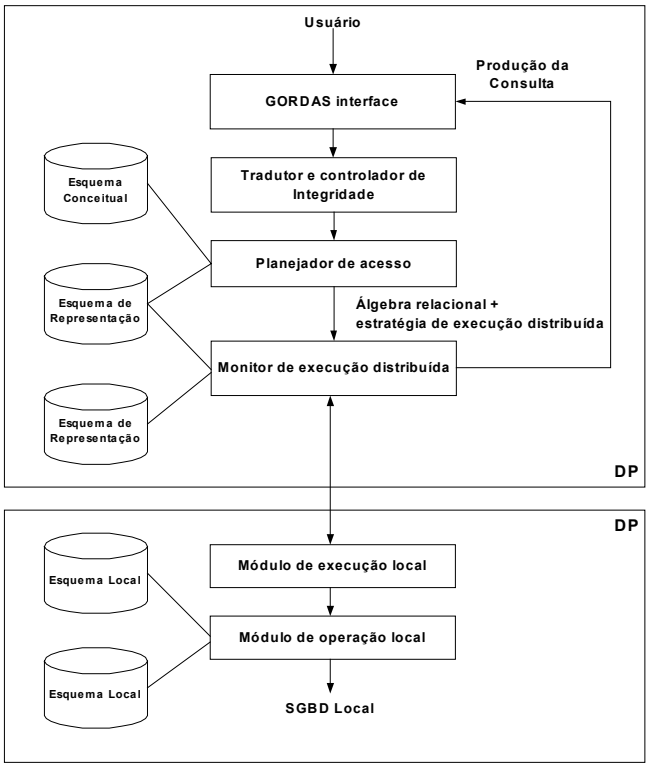

Figura 12 : Componentes de Software do DDTS

O planejador de acesso propõe uma estratégia para um processamento eficiente das aplicações distribuídas. O algoritmo de otimização implementado no DDTS determina o custo mínimo de transmissão selecionando cópias que estão mais próximas do *site* de origem da aplicação. Deste modo a estratégia de custo mínimo é determinada.

O monitor de execução distribuída (DEM) e o monitor de execução local (LEM) cooperam na execução de transações. O DEM cria um conjunto de processos LEM. Cada processo DEM é retido até que a transação é confirmada (*commit*) ou abortada

(*abort*). Os algoritmos utilizados são os 2PC (2-phase-commitment) e 2PL (2-phase-locking). Finalmente, o módulo de operação local traduz e otimiza as subtransações.

2.9. Comentários finais

Na atualidade é difícil conceber uma aplicação que irá manipular informações sobre pessoas e organizações sem a presença de um sistema capaz de gerenciar de maneira eficiente, ágil e segura os dados necessários para esta aplicação. Também é muito comum que muitas organizações, dependendo da natureza da mesma, tenham necessidade de manter os seus dados em várias bases distribuídas por muitas razões, dentre elas podem ser citadas questões geográficas, necessidade de partilhamento, divisão de carga de processamento, maior disponibilidade, entre muitos outros motivos. Assim, a presença de um sistema capaz de gerenciar dados distribuídos é fundamental.

Mas em muitos casos, a necessidade de integração surge depois que os diversos *site*s a serem integrados já possuem uma estrutura local, com seu próprio banco de dados, o que traz a tona uma série de questões a serem resolvidas ao se tentar integrar estes dados legados. Estas situações são muito comuns principalmente com a necessidade de agilidade criada pela popularização da Internet.

Em meio a tudo isto, torna-se necessário a especificação de um modelo capaz de mediar as trocas de informações entre bases heterogêneas, como é o caso do HEROS e do Jupter abordados neste capítulo que possibilitam a integração de bases relacionais heterogêneas através de uma estrutura baseada em mediadores.

3.A Linguagem XML

Este capítulo aborda o padrão XML, sua origem, suas características, funcionalidades, subpadrões e especificações. Também serão mostradas e comparadas as principais linguagens para definição de esquemas para dados XML, as principais linguagens de consulta para dados XML, e a questão da integridade de dados armazenados no formato XML.

3.1. Conceitos básicos

A necessidade de troca de informações entre computadores e sistemas computacionais é viabilizada pela existência de padrões para intercâmbio de dados. O padrão XML é um formato ideal para armazenagem, intercâmbio e posterior publicação de dados estruturados e semi-estruturados através das mais variadas mídias [Bradley, 98]. A sigla XML representa *eXtensible Markup Language* (onde o X substitui o E por questões de estética), suas especificações são mantidas pela W3C[6] e foi desenvolvida baseada em experiências com outras linguagens de marcação.

Um documento XML contem instruções especiais chamadas de *tags*, as quais, identificam o conteúdo do documento. Conforme vemos no exemplo da Lis*tag*em 1, os dados compreendidos entre as *tags* <pessoa> e </pessoa> contém dados referentes a pessoas.

```
<pessoa>
    <nome> Paulo da Silva </nome>
    <idade> 46 </idade>
    <e-mail> paulo@familiasilva.com.br </e-mail>
</pessoa>
```

Lis*tag*em 1 : Exemplo de documento XML contendo dados de uma pessoa.

Em 1996, especialistas em SGML[7], a principal linguagem de marcação da qual surgiu a HTML, sob a chefia de Jon Bosak, da Sun Microsystems, se uniram para definição de um novo padrão de marcação que pudesse ser utilizado na Internet,

[6] O W3C que um grupo que especifica padrões para as tecnologias relacionadas a *Web* (*World Wide Web Consortium* - http://www.w3.org).

[7] A sigla SGML é um padrão ISO (ISO 8879). Esse padrão especifica as regras para a criação de linguagens de marcação indepente da plataforma.

constituindo-se em uma versão simplificada da SGML, cujo objetivo principal era fornecer aos desenvolvedores da *Web* maneiras de definir e criar seus próprios marcadores e atributos quando necessário, em vez de estarem restritos ao esquema de marcação da HTML. No final de 1996, o comitê de trabalho anunciou a primeira versão da XML em uma conferência da SGML, realizada em Boston, nos Estados Unidos. Novos recursos foram consolidados no primeiro semestre de 1997.

A meta principal do comitê foi desenvolver uma linguagem de marcação que tivesse a capacidade e a generalidade da SGML, e fosse fácil de ser implementada na *Web*. Resumidamente, as características desejadas inicialmente para a XML se referiam a três partes:

- a definição da linguagem em si (XML-LANG);
- a definição da ligação entre os documentos (XML-LINK);
- a forma de apresentação dos documentos (XS[8]).

As regras básicas para a criação dessa linguagem de marcação, isto é, as principais características desejáveis para implementação na *Web* eram as seguintes:

- Criar uma linguagem simples, que possibilite a rápida construção de documentos para utilização na *Web*;
- Fornecer suporte à criação de aplicações compatíveis com a abordagem da HTML;
- Possibilitar o desenvolvimento de uma grande variedade de aplicativos, aproveitando-se de seus recursos;
- Fornecer um mecanismo de apresentação genérico e poderoso, permitindo ao desenvolvedor criar a forma de apresentação que mais se adapte às suas necessidades;
- Fornecer suporte para a criação de marcadores personalizados, definidos pelo desenvolvedor do documento *Web*;

[8] A sigla XS aqui significa XML *Stylesheet* ou folhas de estilo.

- Permitir a criação de documentos que pudessem ser validados, isto é, que existisse uma forma de verificar a estrutura do documento, verificando se seus elementos eram válidos, da mesma forma que ocorria com a SGML;
- Fornecer suporte para criação de *hiperlinks* que fossem compatíveis com a especificação de endereços URL, de modo a criar ligações entre documentos;
- Fornecer um mecanismo de folha de estilo genérico e poderoso, que possibilitasse não apenas a formatação do documento, como também sua manipulação.

Uma vez contempladas essas características, a XML passa a fornecer um meio completo para a elaboração e distribuição de documentos por toda a *Web*, sendo independente de plataformas e de sistemas. O objetivo era transformar o conceito da HTML, fornecendo a XML recursos adicionais para a criação e distribuição de documentos.

3.2. Importância da XML

A XML não é apenas mais uma linguagem de marcação como a HTML, pois ela possibilita a utilização de vários recursos importantes. A possibilidade de o desenvolvedor definir marcadores (*tags*) personalizados torna o documento "mais inteligente", dando significado ao texto armazenado entre os marcadores. Esse é o aspecto mais importante da XML.

A XML é independente de plataforma e não é uma linguagem de programação. Ela não faz nada por conta própria e também é de domínio público, constituindo um padrão aberto que nenhuma empresa pode monopolizá-la. Os documentos criados em XML pertencem a seu criador e sua função principal é criar condições para permitir uma padronização na descrição de informações.

Uma vez padronizada a estrutura do documento, é possível, com a utilização de linguagens de programação, interpretar e manipular o conteúdo do documento [Furgeri, 2001].

Um documento XML é composto, basicamente, de três elementos distintos:

- Conteúdo dos dados: são as informações armazenadas entre as *tags*;

- Estrutura: a organização dos elementos dentro do documento, que pode possuir diversos tipos de formato, como um memorando, um contrato, uma receita, um orçamento, enfim, de acordo com as necessidades da marcação da informação;
- Apresentação: é a forma como as informações são apresentadas ao leitor do documento, isto é, como apresentar o conteúdo de um documento XML, pois um mesmo documento pode ser visualizado de forma diferentes.

A idéia central da XML é que muitos benefícios podem ser alcançados quando estes três elementos podem ser mantidos e manipulados de forma separada.

3.3. XML e HTML

Enquanto a HTML indica como algo deve ser exibido, a XML indica o que a informação significa [Furgeri, 2001].

Enquanto a HTML descreve a apresentação dos dados, como tamanho do título ou da fonte que será usado para apresentar os dados em um navegador, a XML descreve o conteúdo destes dados. No exemplo da Lis*tag*em 2, vemos como poderiam ser representados em HTML os dados do exemplo da anterior (Lis*tag*em 1) contendo informações sobre uma pessoa, nele estão presentes apenas *tags* que tratam da apresentação dos dados como por exemplo <h1> que indica que os dados que a compõe representam o cabeçalho, ou a *tag* <p> que indica que o texto deve ser exibido em uma nova linha.

```
<html>
   <h1> Dados de uma Pessoa </h1>
   <p> Nome : Paulo da Silva
   <p> Idade : 46
   <p> e-mail : paulo@familiasilva.com.br
</html>
```

Lis*tag*em 2 : Exemplo de documento HTML contendo dados de uma pessoa.

Segundo [Abiteboul et al., 2000], a XML difere da HTML em três aspectos:

- Novas *tags* podem ser definidas;
- As estruturas podem ser aninhadas e agrupadas sem limite de profundidade;
- Um documento XML pode conter uma descrição opcional da sua gramática.

Documentos XML são ditos como bem formados quando não possuem restrições quanto a marcas, nomes de atributos ou outros padrões, ou seja, quando um documento XML satisfaz uma gramática baseada na qual ele foi especificado ele é considerado válido.

A XML permite a definição de novas *tags* para representar a estrutura dos dados, porém, ao contrário da HTML, não traz nenhuma descrição de como os dados devem ser apresentados. Estas informações sobre a apresentação dos dados deve ser incluídas em separado a uma folha de estilos (*Stylesheet*). As folhas de estilos em uma especificação denominada XML *Stylesheet Language* (XSL) são usadas para converter os dados XML para HTML, podendo assim o seu resultado ser mostrado em um navegador padrão [Abiteboul et al., 2000].

3.4. Componentes de um documento XML

Um documento XML é composto por vários tipos de informações que são usadas para representar os dados presentes no documento e as informações (*metadados*) referentes a estes dados.

3.4.1. Elementos

XML é uma representação textual de dados. Um elemento é formando por uma *tag* inicial, uma *tag* final e os dados compreendidos entre elas. No exemplo da Lis*tag*em 1, os dados compreendidos entre as *tags* <pessoa> e </pessoa> compõe o elemento pessoa. O termo subelemento também é utilizado para representar relações entre um elemento e os elementos que o compõe, assim, podemos dizer que o elemento e-mail (que é composto pelos dados presentes entre as *tags* <e-mail> e </e-mail>) é um subelemento do elemento pessoa.

Na representação de dados XML usam-se elementos repetidos com as mesmas *tags* para representar coleções. O documento XML da Lis*tag*em 3 contem um exemplo no qual várias *tags* <pessoa> aparecem uma após a outra representando assim, dados de uma coleção de pessoas.

```
<tabela>
   <descricao> Artistas Famosos </descricao>
   <pessoas>
      <pessoa>
```

```
         <nome> Raul Seixas </nome>
         <idade> 46 </idade>
         <e-mail> raul@seixas.com.br </e-mail>
      </pessoa>
      <pessoa>
         <nome> Kurt Kobain  </nome>
         <idade> 28 </idade>
         <e-mail> kurt@kobain.com </e-mail>
      </pessoa>
   </pessoas>
</tabela>
```

Lis*tag*em 3 : Exemplo de documento XML representando uma coleção de pessoas.

A XML permite utilização de abreviação na representação de elementos vazios como por exemplo o elemento banda no exemplo da Lis*tag*em 4, pode ser representado apenas pela *tag* <banda/> conforme o descrito na Lis*tag*em 5.

```
<pessoa>
   <nome> Jimi Hendrix </nome>
   <idade> 42 </idade>
   <e-mail> jimi@hendrix.com </e-mail>
   <banda> </banda>
</pessoa>
```

Lis*tag*em 4 : Documento XML sem abreviação de *tags*.

```
<pessoa>
   <nome> Jimi Hendrix </nome>
   <idade> 42 </idade>
   <e-mail> jimi@hendrix.com </e-mail>
   <banda/>
</pessoa>
```

Lis*tag*em 5 : Documento XML com abreviação de *tags*.

3.4.2. Atributos

A XML permite a associação de atributos aos elementos. Os atributos são declarados dentro da *tag* inicial do elemento e são definidos pelo par nome=valor.

Segundo [Bradley, 1998], os atributos servem para representar informações mais refinadas sobre um elemento. No exemplo da Lis*tag*em 6, os atributos são usados para informar o idioma, o ISBN do livro e a moeda corrente.

```
<produto>
   <livro idioma="Ingles" isbn="0-200-987508">
      <titulo> The XML Companion </titulo>
      <preco moeda="Real"> 75.00 </preco>
   </livro>
</produto>
```

Lis*tag*em 6 : Exemplo de utilização de atributos em elementos de um documento XML.

Assim como *tags* os usuários podem definir novos atributos, sendo que o valor dos atributos deve ser sempre uma cadeia de caracteres (*string*) e deve ser representada entre aspas (“).

Existem diferenças entre os atributos e as *tags*. Um atributo pode ocorrer uma vez apenas juntamente com a *tag*, ao passo que subelementos com a mesma *tag* podem ser repetidos. O valor do atributo é sempre uma *string*, enquanto que os dados compreendidos entre uma *tag* inicial e uma *tag* final podem contêr subelementos.

No intercâmbio de dados, os atributos trazem uma certa ambigüidade de quando representar as informações como atributos ou elementos [Abiteboul et al., 2000]. Por exemplo, podemos representar as mesmas informações sobre uma pessoa como na Lis*tag*em 7, ou como na Lis*tag*em 8, ou ainda como na Lis*tag*em 9:

```
<pessoa>
      <nome>Jimi Hendrix</nome>
      <idade>42</idade>
      <e-mail>jimi@hendrix.com</e-mail>
</pessoa>
```

Lis*tag*em 7 : Ambigüidade na representação Elementos X Atributos - 1.

```
<pessoa nome="Jimi Hendrix" idade="42" e-mail="jimi@hendrix.com"/>
```

Lis*tag*em 8 : Ambigüidade na representação Elementos X Atributos - 2.

```
<pessoa idade="42">
   <nome> Jimi Hendrix </nome>
   <e-mail> jimi@hendrix.com </e-mail>
</pessoa>
```

Lis*tag*em 9 : Ambigüidade na representação Elementos X Atributos - 3.

3.4.3. Outros componentes da XML

A linguagem XML possui ainda outros componentes utilizados na criação de documentos utilizados pelas aplicações que são muito pouco, ou nada utilizados no intercâmbio de dados.

3.4.3.1. Comentários

Os comentários são úteis para escrever notas em seus documentos (tanto XML quanto HTML ou em linguagens de programação) para que você saiba porque utilizou determinado elemento ou quando uma parte das informações necessitam de atualização

ou maior atenção. Os comentários em um documento XML são identificados por uma *tag* especial que é aberta pelos caracteres <!-- e fechada pelos caracteres -->. A Lis*tag*em 10 mostra a utilização dos comentários em um documento XML.

```
<pessoa>
      <nome>Jimi Hendrix</nome> <!-- Nome da pessoa -->
      <idade>42</idade>         <!-- Idade da pessoa -->
      <e-mail>jimi@hendrix.com</e-mail> <!-- E-mail da pessoa -->
</pessoa>
```

Lis*tag*em 10 : Utilizando comentários em um documento XML.

3.4.3.2. Instruções de Processamento

As instruções de processamento são definidas por *tags* iniciadas por <? e terminadas por ?> e permitem que o documento contenha instruções que serão executadas pelas aplicações.

Além de declarar a versão da XML, as instruções de processamento também são utilizadas para especificar a folha de estilo a ser usada, entre outras coisas. A declaração XML é opcional, mas se for incluída, deve ser a primeira linha em seu documento conforme o exemplo da Lis*tag*em 11.

```
<?xml version="1.0" ?>
<pessoa>
      <nome>Jimi Hendrix</nome>
      <idade>42</idade>
      <e-mail>jimi@hendrix.com</e-mail>
</pessoa>
```

Lis*tag*em 11 : Instrução de processamento em um documento XML.

Também pode ser necessário utilizar essa instrução de processamento XML inicial para atribuir a codificação de caracteres utilizada no documento, como por exemplo UTF-8 conforme a Lis*tag*em 12, ou outra.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<pessoa>
      <nome>Jimi Hendrix</nome>
      <idade>42</idade>
      <e-mail>jimi@hendrix.com</e-mail>
</pessoa>
```

Lis*tag*em 12 : Declaração do tipo de codificação através de uma instrução de processamento.

3.4.3.3. Seções CDATA

Para inserir um conteúdo em um documento XML e evitar que o analisador (*parser*) interprete este conteúdo pode-se para isto criar uma seção CDATA no documento XML. Uma seção CDATA é representada por uma *tag* aberta pelo conjunto de caracteres <![CDATA[e finalizada por]]>.

Uma característica importante dessa seção, é que não é possível aninhar seções CDATA. Além disso, o início desta seção só pode aparecer depois do início do elemento raiz e o final da seção CDATA também só pode aparecer antes do final do elemento raiz.

A Lis*tag*em 13 nos mostra como uma seção CDATA pode ser utilizada em um documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<pessoa>
      <![CDATA[Este texto não será processado por nenhum
               parser pois o mesmo se encontra dentro de
               uma seção CDATA ]]>
      <nome>Jimi Hendrix</nome>
      <idade>42</idade>
      <e-mail>jimi@hendrix.com</e-mail>
</pessoa>
```

Lis*tag*em 13 : Utilizando uma seção CDATA em um documento XML.

3.5. Estrutura lógica de um documento XML

Um documento XML pode ser representado por uma estrutura em árvore pela sua natureza e características básicas, por exemplo, o documento XML da Lis*tag*em 14 possui a representação em árvore conforme a Figura 13.

```
<bibliography>
  <book>
    <title> Data on the Web </title>
    <author> Abiteboul </author>
    <author> Buneman </author>
    <author> Vianu </author>
    <publisher> M. Kaufmann</publisher>
    <year> 1999 </year>
  </book>
  <book>
     <title> Principles of Distributed Database Systems </title>
     <author> Ozsu </author>
     <author> Valduriez </author>
     <publisher> Prentice Hall </publisher>
     <year> 1999 </year>
  </book>
</bibliography>
```

Lis*tag*em 14 : Documento XML contendo informações bibliográficas.

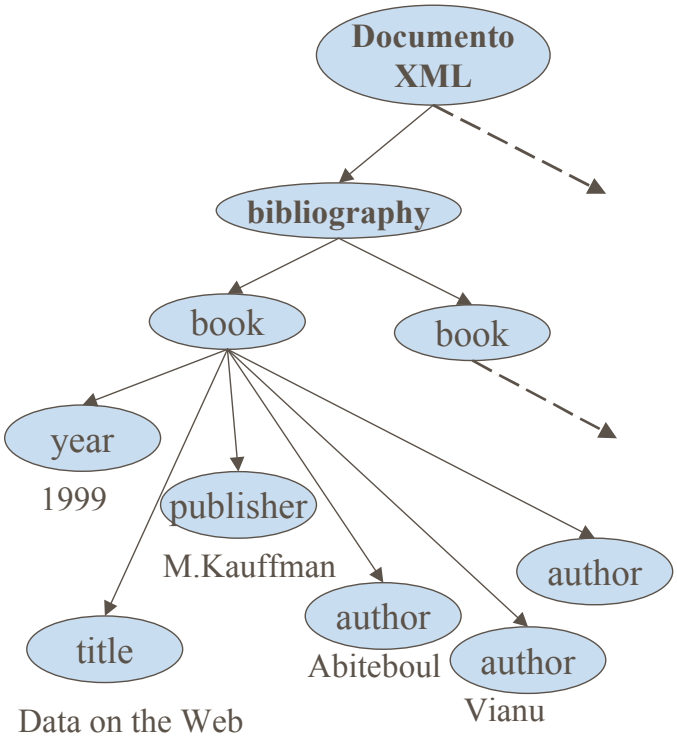

Figura 13 : Estrutura em árvore de um documento XML.

3.5.1. Expressões de Caminho (*Path expressions*)

As expressões de caminho são muito utilizadas para determinar o caminho para se encontrar um elemento dentro de um documento XML tomando como referência e base o elemento raiz.

Um caminho ou *path* é uma seqüência de nós T1.T2.T3. ... .Tn, assim, com uma expressão de caminho podemos encontrar um elemento Tn dentro de um documento XML desde que exista o caminho T1→T2, ..., Tn-1→Tn.

Assim sendo, dada a representação de uma árvore XML correspondente a um documento que armazena informações bibliográficas conforme a Figura 14, podemos determinar por exemplo que:

- conjunto de elementos {b1, b2} está no caminho : biblio.book;
- conjunto de elementos {a1, a2} está no caminho : biblio.book.author;
- conjunto de elementos {a1, t1, a2} está no caminho biblio.book.(author|title).

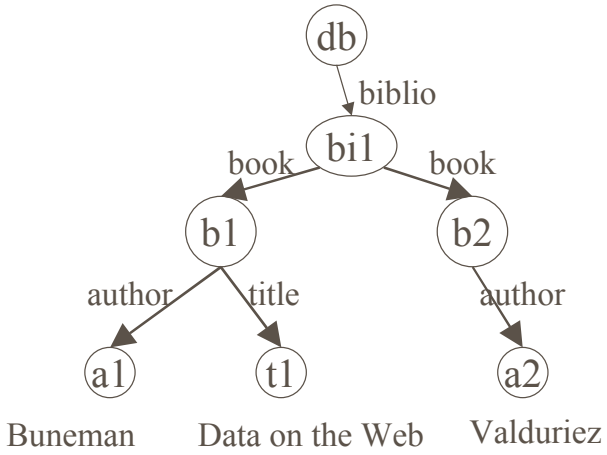

Figura 14 : Arvore XML contendo base de dados bibliográfica.

3.5.2. Xpath

O XPath é um padrão para *path expressions* em XML que utiliza predicados para especificar elementos ou valores de atributos e serve de base para outros padrões XML tais como XSL, entre outros.

A Tabela 3 mostra alguns exemplos de predicados baseados na árvore de dados XML representados na Figura 14:

Predicado	Resultado
/	elemento raiz do documento
/db	um elemento chamado db abaixo (como subelemento) do elemento raiz
db/book	um elemento book imediatamente abaixo do elemento db
db//book	um elemento book em qualquer profundidade
@price	um atributo price
db/book/@price	um atributo price em um elemento book, em db
db/book[@price]	elementos book com um atributo price
db/book[@price=’10’]	elementos book com atributo price igual a 10
//book/para[2]	o 2º parágrafo do conteúdo de qualquer elemento book

Tabela 3 : Exemplos de predicados da linguagem XPath.

3.6. XML e Dados Semi-estruturados

Outro importante tópico de pesquisa em desenvolvimento, no contexto de integração de fontes heterogêneas, é aquele que se refere ao tratamento de dados do tipo semi-estruturados, suas peculiaridades, aplicações e diferentes possibilidades de abordagem.

A partir do momento em que surgiram fontes de dados com características de naturezas diversificadas, como a *Web*, as quais também se deseja tratar e integrar como as bases de dados tradicionais, normalmente representadas por meio de esquemas, ou quando se tornou desejável ter um formato bastante flexível para a troca de dados entre bases de dados diferentes, uma maior atenção sobre dados semi-estruturados se fez necessária[Buneman, 1997].

Várias são as definições apresentadas para dados semi-estruturados. Um conceito bastante aceito define que são dados que não podem ser diretamente representados através dos modelos relacional ou de objetos, podendo ser irregulares ou incompletos [Abiteboul et al., 1998]. Outra definição afirma que este tipo de dado não pode ser representado por qualquer esquema fixo e rígido, apesar de possuir normalmente algum tipo de estrutura implícita a ele associado [Nestorov et al., 1998].

Algumas características próprias dos dados semi-estruturados segundo [Florescu, 1998b] são:

- ausência de esquema previamente definido, o qual pode estar implícito nos próprios dados;
- esquema implícito relativamente grande, passível de alterações freqüentes;
- esquema com função descritiva, em relação ao estado corrente dos dados;
- não definição precisa de tipos de dados, já que para diferentes objetos, por exemplo, os valores de um mesmo atributo podem ser de diferentes tipos, em determinadas situações.

3.7. Linguagens para Esquemas

Vários tem sido os esforços no sentido de se criar mecanismos para se representar a estrutura de um documento XML. Na seqüência serão abordadas as principais propostas de linguagens para definição de esquemas para documentos XML.

3.7.1. Document Type Definition (DTD)

A DTD funciona como uma gramática para o documento XML e é parte da linguagem XML podendo também ser um esquema para os dados representados pelo documento XML, mas para definição destes esquemas, a DTD deixa a desejar e outras propostas surgiram após a adoção do XML como um padrão.

O principal conceito dentro da XML segundo [Mello et al., 2000] é o conceito de elemento, que descreve uma unidade atômica ou não-atômica de dado. Um ou mais elementos podem estar definidos previamente através de uma DTD, que define um padrão para marcação de dados em documentos através da definição de uma hierarquia de elementos, onde um elemento é a raiz desta hierarquia. Para que um documento XML esteja de acordo com uma DTD, apenas os elementos e as estruturas de aninhamento entre elementos definidas na DTD são permitidos no corpo do documento - esta validação é feita por *parsers* XML.

Em uma DTD, uma definição de elemento pode ser atômica ou complexa. No primeiro caso, o elemento possui apenas um conteúdo textual. No segundo caso, o elemento agrega subelementos componentes. Elementos componentes podem ser definidos como obrigatórios ou opcionais e também podem se repetir. Elementos podem ainda ter atributos. Atributos devem pertencer a um tipo de dado[9] e podem ter um valor *default*.

Considere o documento da Lis*tag*em 15:

```
<?xml version="1.0"?>
<base-de-dados>
    <pessoa>
        <nome> Jimi Hendrix </nome>
        <idade> 33 </idade>
        <e-mail> jimi@hendrix.com </e-mail>
    </pessoa>
</base-de-dados>
```

Lis*tag*em 15 : Documento XML para definição de sua estrutura através de esquemas.

[9] DTDs suportam alguns tipos de dados derivados de strings.

A DTD para ela ficaria definido conforme a Lis*tag*em 16:

```
<DOCTYPE base-de-dados [
    <!ELEMENT base-de-dados (pessoa*)>
    <!ELEMENT pessoa (nome, idade, e-mail)>
    <!ELEMENT nome (#PCDATA)>
    <!ELEMENT idade (#PCDATA)>
    <!ELEMENT e-mail (#PCDATA)>
]>
```

Lis*tag*em 16 : DTD para documento da Lis*tag*em 15.

A primeira linha informa que o elemento raiz é o <base-de-dados>. As próximas cinco linhas são declaram que o elemento <base-de-dados> deve conter um número qualquer de elementos "pessoa" cada um contendo os subelementos "nome", "idade" e "e-mail", sendo que os últimos contem apenas dados e não contem outros elementos. Assim, pessoa* é uma expressão regular, indicando que existirão qualquer número de ocorrências do elemento "pessoa". Outras expressões regulares permitidas são e+ (uma ou mais ocorrências), e? (zero ou uma ocorrência), entre outras.

A DTD é uma gramática para o documento XML. No exemplo de DTD da Lis*tag*em 16, podemos notar que o mesmo define que obrigatoriamente os subelementos "nome", "idade" e "e-mail" deverão aparecer nesta ordem no elemento "pessoa".

3.7.2. XML Schema

XML Schema é uma proposta da W3C [Fallside, 2000] para descrever a estrutura de um documento XML. XML Schema é um padrão mais abrangente que uma DTD, permitindo expressar tipos de dados, herança, tipos abstratos, unicidade e chaves, entre outras funcionalidades [Mello et al., 2000].

Uma especificação em XML Schema sempre inicia com a *tag* <*schema*> e termina com a *tag* </*schema*>. Todas as declarações de elementos, atributos e tipos devem ser inseridas entre estas duas *tags*. Tipos, que representam a estrutura de uma classe de documentos e seus relacionamentos com outras classes, podem ser definidos. Um tipo pode ser:

- simples (*simpleType*): é um tipo básico como *string*, *date*, *float*, *double*, *timeDurations*, etc.

- complexo (*complexType*): define a estrutura de um elemento, ou seja, define características como subelementos, atributos, cardinalidades dos subelementos e obrigatoriedade dos atributos.

Considere novamente o documento XML da Lis*tag*em 15, a especificação XML Schema para ela ficaria definido conforme a Lis*tag*em 17:

```
<?xml version="1.0"?>
  <schema
       xmlns:xsd="http://www.w3.org/1999/XMLSchema"
       xmlns:grp="http://meunamespace.com/base-de-dados_pessoas"
       targetNamespace="http://meunamespace.com/exemplobase-de-dados">

  <complexType name="tbase-de-dados">
    <element ref="pessoa" minOccurs='1' maxOccurs='*'/>
  </complexType>

  <complexType name="tpessoa">
    <element name="nome" type="string" minOccurs='1' maxOccurs='1'/>
    <element name="idade" type="integer" minOccurs='1' maxOccurs='1'/>
    <element name="e-mail" type="string" minOccurs='1' maxOccurs='1'/>
  </complexType>

  <element name="base-de-dados" type="tbase-de-dados"/>
  <element name="pessoa" type="tpessoa"/>
</schema>
```

Lis*tag*em 17 : XML Schema para documento da Lis*tag*em 15.

O exemplo acima mostra a declaração de um *complexType* tbase-de-dados e mais adiante uma declaração:

```
<element name="base-de-dados" type="tbase-de-dados"/>
```

Esta segunda declaração liga o elemento base-de-dados ao *complexType* tbase-de-dados, indicando que, em uma instância de um documento XML que segue este esquema, deve-se ter um elemento base-de-dados com o subelementos pessoa, que da mesma forma está ligado ao *complexType* tpessoa que deve conter os elementos nome, idade e e-mail. As cardinalidades mínima e máxima são indicadas pelos atributos *MinOccurs* e *MaxOccurs*, respectivamente.

ComplexTypes podem ter atributos, que são declarados através da *tag* <*attribute*> e devem ser do tipo *simpleType*. Um atributo pode ser declarado como obrigatório ou opcional através da cláusula *use*. Os valores permitidos para esta cláusula são *required* (obrigatório), *optional* (opcional) ou *fixed* (fixo). No último caso, deve-se dizer o valor default do atributo utilizando a cláusula value.

Ainda, pode-se restringir o conteúdo de um elemento através do uso de um atributo chamado *content*, que pode assumir os seguintes valores: *textOnly* (apenas texto); *elementOnly* (apenas subelementos); *mixed* (texto e subelementos); ou *empty* (conteúdo vazio).

A XML Schema possui um mecanismo de derivação de tipos, permitindo a criação de novos tipos a partir de outros já existentes. Isto pode ser feito de duas maneiras: por restrição ou por extensão. Tipos simples só podem ser derivados por restrição, aplicando-se "facetas" a um tipo básico ou utilizando uma linguagem de expressões regulares [Mello et al., 2000]. O exemplo abaixo mostra a derivação de um tipo simples chamado inteiroDerivado através da aplicação de facetas que restringem o valor mínimo e máximo de um valor inteiro.

```
<simpleType name="inteiroDerivado" base="integer">
        <minInclusive value="1"/>
        <maxInclusive value="20"/>
</simpleType>
```

Lis*tag*em 18 : Derivação de tipos com a XML Schema.

Tipos complexos podem ser derivados por restrição ou por extensão. A derivação por restrição permite, por exemplo, restringir a cardinalidade de um subelemento. A derivação por extensão adiciona características a um tipo, sendo semelhante ao conceito de herança [Fallside, 2000]. O exemplo da Lis*tag*em 19 mostra o *complexType* tpessoacomfiliacao acrescenta os elementos "nome-do-pai" e "nome-da-mae" ao *complexType* tpessoa, com cardinalidades mínima igual a 1 e máxima igual a 1.

```
<complexType name="tpessoacomfiliacao" base="tpessoa"
derivedBy="extension">
    <element name="nome-do-pai" type="string" minOccurs='1'
maxOccurs='1'/>
    <element name="nome-da-mae" type="string" minOccurs='1'
maxOccurs='1'/>
</complexType>
```

Lis*tag*em 19 : Derivação de um complexType em XML Schema.

Grupos especificam restrições sobre um conjunto fixo de subelementos, podendo ser de três tipos:

- *sequence* estabelece que todos os elementos pertencentes a ele devem aparecer na ordem em que foram definidos e nenhum pode ser omitido;

- *choice* estabelece que apenas um dos elementos pertencentes ao grupo deve aparecer em uma instância XML;
- *all* estabelece que os elementos podem aparecer em qualquer ordem e podem ser repetidos ou omitidos.

Um exemplo de definição de um grupo do tipo sequence para o *complexType* tpessoa pode ser visto na Lis*tag*em 20.

```
<group>
  <sequence>
      <complexType name="tpessoa">
          <element name="nome" type="string" minOccurs='1'
maxOccurs='1'/>
          <element name="idade" type="integer" minOccurs='1'
maxOccurs='1'/>
          <element name="e-mail" type="string" minOccurs='1'
maxOccurs='1'/>
      </complexType>
  </sequence>
</group>
```

Lis*tag*em 20 : Definição de grupos com XML Schema.

Uma declaração de atributo, elemento ou grupo pode ser referenciada, permitindo a reutilização de declarações, como a declaração:

```
<element ref="pessoa" minOccurs='1' maxOccurs='1'/>
```

dentro do *complexType* tbase-de-dados na Lis*tag*em 17. A única restrição no uso de referências é que o elemento referido seja global, ou seja, tenha sido declarado dentro de <*schema*>, porém, não dentro de um *complexType*.

3.7.2.1. *Namespaces*

Pelo fato da XML ser extensível com a possibilidade da criação de qualquer *tag*, conflitos entre estas *tags* podem ocorrer. Num mesmo documento podem existir dois elementos com mesmo nome, mas que possuam significados diferentes. Ou ainda, como não existe um órgão global responsável pela especificação de elementos XML (isto tiraria a flexibilidade da linguagem), um programador pode criar uma *tag* para armazenar determinada informação e, outro programador pode criar uma outra *tag* para guardar o mesmo tipo de informação. Por exemplo, um dono de uma vídeo locadora cria a *tag* GÊNERO para indicar se o filme é de comédia, terror, ação ou drama. Já um outro

dono de locadora, cria a *tag* TIPO para o mesmo objetivo. Se as duas empresas forem trocar informações, ocorrerá um conflito ou não entendimento dos dados contidos no documento XML.

Para esta problemática foi criada uma pequena extensão para XML denominada *namespaces*. *Namespaces* não limitam a facilidade de extensão da XML, mas insere um mecanismo para gerenciá-la. Com os *namespaces* pode-se misturar elementos descritivos definidos por comunidades independentes, sem medo de cometer discrepâncias na nomeação, uma vez que cada segmento de dados está vinculado a um URI *(Universal Resource Indicato*r) que fornece um contexto e uma definição para tal segmento.

Um esquema especificado através do XML Schema pode ser visto como um conjunto de declarações de tipos e elementos cujos nomes pertencem a um *namespaces* [Bray et al., 1999]. Todo esquema definido através de XML Schema deve definir um único namespace, sendo usual o formato de URI para a sua identificação

O uso de *namespaces* aumenta a flexibilidade de XML Schema, permitindo a reutilização de definições feitas em outros esquemas. A utilização de um tipo definido em outro esquema é possível através da sua declaração e da associação de um prefixo a ele [Mello et al., 2000].

```
<?xml version="1.0"?>
<referencias xmlns:mau="http://mauri.org/documents/">
    <mau:descricao>Artigo sobre P2P</mau:descricao>
    <mau:autor>Mauri Ferrandin </mau:autor>
    <mau:data>15/07/2002</mau:data>
</referencias>
```

Lis*tag*em 21 : Utilização de *namespaces* em XML

Na Lis*tag*em 21, foram utilizados elementos definidos na URI mauri.org e foi utilizado como *namespace* o prefixo "mau", que, por sua vez, está associado a um URI. Neste URI é que estão definidos o que significa cada elemento *(*autor, data, descricao, etc.). Com o recurso do *namespac*e, é permitido que cada autor elabore semânticas adicionais requeridas pelos tipos particulares de recursos ou por uma área de atuação específica.

3.7.3. XDR

Primeiramente chamada de XML-Data e mais tarde abreviada para XDR (XML-Data Reduced), esta linguagem é um esforço da Microsoft e algumas outras empresas e foi usada no Microsoft Bizz Talk Framework. A XDR foi fortemente influenciada por outra proposta desenvolvida em cooperação entre a Microsoft e a IBM denominada DCD (Document Content Description) o que faz com que as duas tenham muitas características em comum.

3.7.4. SOX

A SOX (Schema for Objetc-Oriented XML) é uma linguagem alternativa para definição da estrutura sintática de um documento XML e parcialmente a sua estrutura semântica. Conforme o próprio nome, ela é extensão da DTD incorporando funções de orientação a objetos, permitindo a herança de tipos de dados e de elementos. É mantida e desenvolvida pela Commerce One.

3.7.5. Schematron

O Schematron foi criada por Rick Jelliffe, é um pouco diferente das demais uma vez que seu foco está na validação de esquemas usando padrões ao invés de definir esquemas. Seu esquema é simples e pode ser representado em uma simples página, além de permitir a definição de restrições poderosas através da linguagem XPath.

3.7.6. DSD

A DSD foi desenvolvida pela AT&T Labs em cooperação com a BRICS com o objetivo de ser uma descrição para os elementos e atributos dependendo do contexto, com mecanismos padrões de inserção bem flexíveis e um poder expressivo quando combinada com XSLT. Assim como a Schematron, a DSD é fortemente focada na definição de esquemas baseados em restrições.

3.7.7. Tabela comparativa

A Tabela 4 apresenta um comparativo entre as seis principais linguagens para definição de esquemas de dados para XML segundo [Lee e Chu, 2000].

Features	DTD 1.0	XML Schema 1.0	XDR 1.0	SOX 2.0	Schematron 1.4	DSD 1.0
Esquemas						
Sintaxe em XML	Não	Sim	Sim	Sim	Sim	Sim
Namespace	Não	Sim	Sim	Sim	Sim	Não
Inclusões	Não	Sim	Não	Sim	Não	Sim
Importações	Não	Sim	Não	Sim	Não	Não
Tipos de Dados						
Tipos pré-definidos	10	37	33	17	0	0
Tipos definidos pelo usuário	Não	Sim	Não	Sim	Não	Sim
Restrições de domínio	Não	Sim	Não	Parcial	Sim	Sim
Tipos nulos	Não	Sim	Não	Não	Não	Não
Atributos						
Valor default	Sim	Sim	Sim	Sim	Não	Sim
Tipo escolha (choice)	Não	Não	Não	Não	Sim	Sim
Opcional vs. Obrigatório	Sim	Sim	Sim	Sim	Sim	Sim
Restrições de domínio	Parcial	Sim	Parcial	Parcial	Sim	Sim
Definições condicionais	Não	Não	Não	Não	Sim	Sim
Elementos						
Valor default	Não	Parcial	Não	Não	Não	Sim
Modelo de conteúdo	Sim	Sim	Sim	Parcial	Sim	Sim
Seqüência ordenada	Sim	Sim	Sim	Sim	Sim	Sim
Seqüência não ordenada	Não	Sim	Sim	Não	Sim	Sim
Escolha	Sim	Sim	Sim	Sim	Sim	Sim
Ocorrências máx. e mín.	Parcial	Sim	Sim	Sim	Sim	Parcial
Modelo aberto	Não	Não	Sim	Não	Sim	Não
Definição condicional	Não	Não	Não	Não	Sim	Sim
Herança						
Tipos simples por extensão	Não	Não	Não	Não	Não	Não
Tipos simples por restrição	Não	Sim	Não	Sim	Não	Não
Tipos complexos por extensão	Não	Sim	Não	Sim	Não	Não
Tipos complexos por restrição	Não	Sim	Não	Não	Não	Não
Unicidade de chaves						
Unicidade para atributos	Sim	Sim	Sim	Sim	Sim	Sim
Unicidade para não atributos	Não	Sim	Parcial	Não	Sim	Não
Chave para a atributos	Não	Sim	Não	Não	Sim	Não
Chave para não atributos	Não	Sim	Não	Não	Sim	Não
Chave estrangeira para a atributos	Parcial	Sim	Parcial	Parcial	Sim	Sim
Chave estrangeira para não atributos	Não	Sim	Não	Não	Não	Sim
Outros						
Restrições dinâmicas	Não	Não	Não	Não	Sim	Não
Versões	Não	Não	Não	Não	Não	Sim
Documentação	Não	Sim	Não	Sim	Sim	Sim
HTML Embutido	Não	Sim	Não	Sim	Parcial	Sim
Auto descrição	Não	Parcial	Não	Não	Parcial	Sim

Tabela 4 : Comparativo entre as seis principais linguagens para esquema.

3.8. Linguagens de consulta para dados XML

Uma das grandes questões é por que não adaptar a SQL ou a OQL para executar consultas sobre dados XML. A resposta esta no fato de que dados XML são fundamentalmente diferentes de dados relacionais ou orientados a objetos, e então, nem a SQL, nem a OQL são apropriadas para consultas a dados XML.

O fator chave que diferencia dados XML dos dados armazenados em modelos tradicionais é que os dados XML não apresentam estrutura rígida. Nos esquemas relacionais e orientados a objetos, os dados possuem um esquema que esta separado e é independente dos dados, já no caso de dados XML o esquema está juntamente com os dados. Assim, podemos dizer que os dados XML são auto-descritivos e podem naturalmente modelar dados com estruturas irregulares e que não podem ser modelados em esquemas tradicionais.

3.8.1. Requisitos de uma linguagem de consulta para dados XML

Nesta seção, serão abordados os requisitos necessários para uma linguagem de consulta para dados XML.

- Precisão semântica. Uma linguagem de consulta para dados XML deve possuir uma formalidade semântica.
- Habilidade de rescrita e otimização. Os dados XML serão gerados a partir de outros formatos, tais como relacional e orientado a objetos, ou outros formatos utilizados para propósitos mais específicos.
- Operações de consulta. As diferentes operações que devem ser suportadas por uma linguagem de consulta são: seleção, extração, redução, reestruturação, combinação
- Semântica composicional. As expressões definidas na linguagem de consulta XML devem possuir transparência referencial, ou seja, o significado de uma expressão deve ser o mesmo independente de onde ela aparece.
- Não requer esquema. Uma linguagem de consulta para dados XML deve executar consultas a dados XML mesmo quando não existir um esquema definido para estes dados (DTD, XML-Schema ou outro).

- Explora esquema disponível. Quando o DTD estiver disponível para a fonte de dados, a linguagem de consulta deve ser capaz de julgar quando a consulta está corretamente formulada em relação ao DTD.
- Preserva ordem e a associação. Uma linguagem de consulta para dados XML deve ser capaz de preservar a ordem e a associação dos elementos dos dados XML, se necessário.
- Baseada em XML. Uma consulta XML deve ser capaz de conter dados arbitrários XML, e um documento XML deve ser capaz de armazenar consultas arbitrárias.
- Suporta novos tipos de dados. Uma linguagem de consulta para dados XML deve conter um mecanismo de extensão para operações e condições específicas para um tipo de dado em particular.
- Apropriada para metadados. A linguagem de consulta para dados XML deve poder ser utilizada como parte da descrição dos metadados.
- Processada no servidor. Uma linguagem de consulta para dados XML deve apresentar a possibilidade de ser executada remotamente no servidor independente do contexto local da aplicação.
- Manipulável através de programação. Uma consulta poderá ser formulada por programas em tempo de execução.
- Representável através de XML. Uma consulta deve ser representável através de XML.

3.8.2. Exemplo de linguagem de consulta para dados XML (XML-QL)

Na seqüência serão demonstradas alguns casos de uso, características e funcionalidades de uma linguagem para consulta a dados XML. A linguagem será a XML-QL, uma vez que a mesma será utilizada na etapa de desenvolvimento do protótipo proposto por este trabalho.

Tomemos um documento XML com a estrutura descrita de acordo com a DTD definida na Lis*tag*em 22 disponível em um determinado *site* acessível através da URI www.a.b.c/bib.xml:

```
<!ELEMENT livro (autor+, titulo, editora)>
  <!ATTLIST livro ano CDATA>
  <!ELEMENT artigo (autor+, titulo, ano?,
(versaoresumida|versaocompleta))>
  <!ATTLIST artigo tipo CDATA>
  <!ELEMENT editora (nomeeditora, endereco)>
  <!ELEMENT autor (nome?, sobrenome)>
```

Lis*tag*em 22 : DTD para exemplo de consultas XML-QL.

Esta DTD especifica que um elemento "livro" contém um ou mais elementos "autor", um elemento "titulo", um elemento "editora" e um atributo "ano". Um artigo é similar, sendo que o atributo ano é opcional, não possui "editora", e possui um elemento "versaoresumida" ou um elemento "versaocompleta". Um elemento "artigo" possui um atributo chamado "tipo". O elemento "editora" contém um elemento "nomeeditora" e um elemento endereço, e um elemento autor possui um elemento "nome" não obrigatório e um elemento sobrenome obrigatório. Assumimos que os campos "nomeeditora", "endereço", "nome" e "sobrenome" são todos do tipo CDATA.

Recuperando dados através da identificação de padrões : a XML-QL usa elementos padrões para recuperar dados de um documento XML. O exemplo de consulta da Lis*tag*em 23 retorna todos os autores que publicaram obras pela editora "Addison-Wesley" que estão presentes em um documento XML na URI www.a.b.c/bib.xml. Toda a URI que representa uma fonte de dados XML deve aparecer a direita da palavra reservada IN.

```
WHERE <livro>
        <editora>
            <nomeeditora>Addison-Wesley</nomeeditora>
        </editora>
        <titulo> $t</titulo>
        <autor> $a</autor>
</livro> IN "www.a.b.c/bib.xml"
CONSTRUCT $a
```

Lis*tag*em 23 : Exemplo de consulta básica XML-QL.

Na prática, esta consulta encontra todos os elementos <livro> no documento XML presente na URI www.a.b.c/bib.xml que tenha um subelemento <titulo>, um subelemento <autor> e um subelemento <editora>, sendo que o último possui um

subelemento <nomeeditora> o qual possui valor igual a “Addison-Wesley”. Para cada elemento que se enquadre na consulta ele armazena o nome do autora na variável a ($a) e o título da obra na variável t ($t) . Note que todos os nomes de variáveis são precedidos pele caracter “$” para distinguir as mesmas dos outros valores literais presentes no documento.

Pode-se abreviar o fechamento da *tag* de cada elemento (</elemento> por exemplo) utilizando apenas a *tag* </>. Assim, a consulta anterior (Lis*tag*em 23) pode ser descrita conforme na Lis*tag*em 24.

```
WHERE <livro>
        <editora><nomeeditora>Addison-Wesley</></>
        <titulo> $t</>
        <autor> $a</>
</> IN "www.a.b.c/bib.xml"
CONSTRUCT $a
```

Lis*tag*em 24 : Exemplo de consulta básica XML-QL com abreviação de *tags*.

Montando os resultados em um documento XML: a consulta exibida anteriormente retorna a lista de todos os autores com publicações na editora “Addison-Wesley” presentes no documento. É muito importante porém que os resultados de uma consulta XML-QL sejam retornados no formato XML por questões de padronização.

A consulta na seqüência retorna todos os nomes dos autores e os títulos de suas publicações dentro de um elemento raiz chamado <resultado> :

```
WHERE <livro>
        <editora><nomeeditora>Addison-Wesley</></>
        <titulo> $t</>
        <autor> $a</>
    </> IN "www.a.b.c/bib.xml"
CONSTRUCT
          <resultado>
                  <autor> $a </>
                  <titulo> $t </>
          </>
```

Lis*tag*em 25 : Consulta XML-QL formatando os resultados em XML.

Para exemplificar melhor, tomemos o documento XML da Lis*tag*em 26:

```
<bib>
    <livro ano="1995">
        <titulo> Integrando fontes de dados</titulo>
        <autor> <sobrenome> Ferrandin</sobrenome> </autor>
        <editora>
            <nomeeditora> Addison-Wesley </nomeditora>
        </editora>
    </livro>
    <livro ano="1998">
```

```
        <titulo> Foundation for Object/Relational Databases
        </titulo>
        <autor> <sobrenome> Date </sobrenome> </autor>
        <autor> <sobrenome> Darwen </sobrenome> </autor>
        <editora>
            <nameeditora> Addison-Wesley </nameeditora >
</editora>
    </livro>
</bib>
```

Lis*tag*em 26 : Documento XML contendo dados bibliográficos.

Aplicando a consulta anterior (Lis*tag*em 25) sobre o documento exemplo acima (Lis*tag*em 26) obteremos os resultados conforme a Lis*tag*em 27:

```
<resultado>
    <autor><sobrenome> Date </sobrenome> </autor>
    <titulo>Na Introduction to Database Systems </titulo>
</resultado>
<resultado>
    <autor> <sobrenome> Date </sobrenome> </autor>
    <titulo>Foundation for Object/Relational
            Databases</titulo>
</resultado>
<resultado>
       <autor> <sobrenome> Darwen </sobrenome> </autor>
       <titulo>Foundation for Object/Relational
               Databases </titulo>
</resultado>
```

Lis*tag*em 27 : Formatando o resultado de uma consulta XML-QL em XML.

Agrupando dados usando consultas aninhas: a consulta anterior (Lis*tag*em 25) não agrupava os autores de acordo com o livro, ou seja, no caso da existência de dois autores para o mesmo título ela retorna um elemento <resultado> para cada autor. Para agruparmos o resultado por título teremos que usar consultas aninhadas (Lis*tag*em 28) que retornará um elemento <resultado> para cada título e cada um contendo a lista dos seus respectivos autores.

```
WHERE <livro> $p </> IN "www.a.b.c/bib.xml",
          <titulo> $t </>,
          <editora><nomeeditora>Addison-Wesley</>
      </> IN $p
CONSTRUCT <resultado>
             <titulo> $t </>
             WHERE  <autor> $a </> IN $p
             CONSTRUCT <autor> $a</>
          </>
```

Lis*tag*em 28 : Agrupando dados através de consultas aninhadas.

Aplicando esta consulta (Lis*tag*em 28) sobre o documento XML exemplo da Lis*tag*em 26 teremos os resultados mostrados na Lis*tag*em 29:

```
<resultado>
   <titulo> An Introduction to Database Systems </titulo>
   <autor> <sobrenome> Date </sobrenome> </autor>
</resultado>
<resultado>
   <titulo> Foundation for Object/Relational Databases
           </titulo>
   <autor> <sobrenome> Date </sobrenome> </autor>
   <autor> <sobrenome> Darwen </sobrenome> </autor>
</resultado>
```

Lis*tag*em 29 : Resultado de uma consulta agrupando o resultado.

Junções de elementos pelo valor : XML-QL pode expressar junções (joins) encontrando um ou mais elementos que contém o mesmo valor. Por exemplo, a consulta da Lis*tag*em 30, recupera todos os artigos que tenham pelo menos um autor que tenha sido escrito um livro desde 1995. Aqui, assumimos que o autor tem o mesmo nome e sobrenome (representados pelas variáveis $f e $1) para livros e artigos.

```
WHERE <artigo>
        <autor>
          <nome> $f </>
          <sobrenome> $l </>
        </>
      </> CONTENT_AS $a IN "www.a.b.c/bib.xml"
      <livro ano=$y>
        <autor>
          <nome> $f </>
          <sobrenome> $l </>
        </>
      </> IN "www.a.b.c/bib.xml",
      y > 1995
CONSTRUCT <artigo> $a </>
```

Lis*tag*em 30 : Junções de elementos pelo valor em uma consulta XML-QL.

Existem muitas outras consultas possíveis utilizando a linguagem XML-QL, mas a intenção aqui é mostrar apenas o básico. Construções utilizando *tag* com variáveis, expressões regulares de caminhos (regular path expressions), transformações de dados através de consultas XML-QL, integração e consultas a várias fontes simultâneas, criação de funções, *namespaces*, semi-junções entre outras podem ser utilizadas de acordo com as necessidades das aplicações/desenvolvedores.

A gramática completa contendo a BNF[10] da linguagem XML-QL pode ser visualizada junto ao apêndice deste trabalho no Anexo 1 : XML-QL Grammar.

3.8.3. Outras linguagens de consultas para dados XML

Existem várias propostas de linguagens de consulta para dados XML, sendo que serão abordadas aqui as principais, uma lista completa pode ser obtida em http://www.w3.org/TandS/QL/QL98/.

LORE(*Lightweight Object Repository*) é um sistema de gerenciamento de dados semi-estruturados. Sua linguagem de consulta, a LOREL, foi obtida através da extensão da OQL para consultar dados semi-estruturados. Recentemente, a Lorel foi adaptada para consultar dados XML.

A XSL proposta pela W3C pode também ser vista como uma linguagem de consulta para dados XML, mas suas funcionalidades são muito limitadas. Ela não suporta *joins* ou funções do tipo *skolen*, mas ainda assim, pode ser usada como uma linguagem de consulta dentro de um escopo limitado. Ao contrário de outras linguagens, com a XSL, é fácil fazer processamento recursivo. Por exemplo, a consulta XSL da Lis*tag*em 31 recupera todos os elementos autor de um documento, independente da profundidade na qual o mesmo se encontra.

```
<xsl:template> <xsl:apply-templates/> </xsl:template>
      <xsl:template match=''author''>
      <result> <xsl:value-of/> </result>
</xsl:template>
```

Lis*tag*em 31 : Consulta XSL para dados XML.

A XQL é uma linguagem que essencialmente consiste de uma busca de padrões através de XSL com uma sintaxe bem definida para a construção dos resultados. Seu poder de reestruturação é restrito a um subconjunto da linguagem XSL.

A XML-GL é uma linguagem de consulta gráfica para XML, da mesma linha da linguagem QBE[11]. Na XML-GL, ambas as cláusulas WHERE e CONSTRUCT são especificadas através de uma interface gráfica, e suas funcionalidades são similares a XML-QL.

---

[10] BNF é uma notação para descrição formal de linguagens de programação.
[11] QBE (*Query* by Example) – Consultas através de exemplos.

A Tabela 5 mostra um comparativo entre as seis principais linguagens de consulta para dados XML segundo [Bonifati e Ceri, 1999].

Linguagem / Característica	**LOREL**	**XML-QL**	**XML-GL**	**XSL**	**XQL**
Modelos de dados específicos	Sim	Sim	Sim	Sim	Não
Gerenciamento diferencial de IDREFs	Sim	Não	Não	Não	Não
Seleção de Documentos	Sim	Sim	Sim	Sim	Sim
Junções	Sim	Sim	Sim	Não	Não
Formato do resultado	Conj. de OIDs	Documento XML	Documento XML	Documento XML	Documento XML
Especificação e expressões de caminho parciais.	Sim	Sim	Parcial	Sim	Sim
Parada Quando encontrar dados cíclicos	Sim	Indefinido	Sim	Não	Indefinido
Quantificação existencial	Sim	Sim	Sim	Sim	Sim
Qualificadores Universais	Sim	Não	Não	Não	Sim
Negações	Sim	Não	Sim	Sim	Sim
Reduções	Não	Não	Não	Não	Não
Construção de novos elementos	Sim	Sim	Sim	Sim	Não
Construções em grupos	Sim	Não	Sim	Não	Não
Funções skolem	Sim	Sim	Parcial	Não	Não
Agregação	Sim	Não	Sim	Parcial	Parcial
Consultas aninhadas	Sim	Sim	Não	Sim	Não
Consultas binárias	Sim	Parcial	Sim	Sim	Sim
Ordenação dos resultados	Sim	Sim	Sim	Sim	Não
Preservando ordem dos resultados	Sim	Sim	Sim	Sim	Sim
Consultas ordenadas por esquemas	Não	Sim	Não	Não	Não
Consultas por ordem de instâncias	Sim	Não	Não	Não	Sim
Abstração de tipos	Sim	Não	Não	Não	Não
Coerção de tipos	Sim	Não	Não	Não	Partially
Suporte a RDF	Não	Não	Não	Não	Não
Suporte a Xpointer e Xlink	Não	Não	Não	Não	Não
Tags variáveis	Sim	Sim	Não	Não	Não
Linguagens de atualização	Sim	Não	Sim	Não	Não

Tabela 5 : Comparativo entre linguagens de consulta para XML

3.9. APIs para XML

Existem dois tipos principais de API para dados XML:

- API baseada em árvores (Tree-based APIs) : mapeia o documento XML em uma estrutura de árvore e permite a aplicação navegar entre os nodos desta árvore. O grupo de trabalho para definição do DOM coordenado pela W3C mantém as recomendações das API baseadas em árvore para documentos XML e HTML, sendo que existem muitas outras APIs definidas por outras entidades. O DOM será abordado na seção 3.9.2;
- API baseada em eventos (Event-based API) : reporta eventos no processo de *parsing* diretamente a aplicação através de *callbacks*, e não mapeia os dados para uma estrutura interna de árvore. A aplicação por sua vez implementa os *handlers* para lidar com cada um dos eventos que possam ocorrer. SAX é o melhor exemplo deste tipo de API e será abordado na seção 3.9.1.

As APIs baseadas em árvores são muito úteis para uma grade variedade de aplicações, mas normalmente demandam grande quantidade de recursos do sistema, especialmente se o documento for grande. As APIs baseadas em eventos provêm um acesso simples e de baixo nível ao documento XML, assim, é possível manipular documentos muito maiores que a capacidade de memória do sistema, além de possibilitar a implementação de novas estruturas através do tratamentos de eventos que disparam os *callbacks*.

Considere, por exemplo, a seguinte tarefa : Localizar o elemento que contém a palavra Florianópolis. Se o seu documento for de uns 20MB por exemplo, será ineficiente construir e armazenar na memória uma árvore contendo os dados do arquivo somente para encontrar uma parte muito pequena compreendida no mesma. Utilizando uma API baseada em eventos, será possível encontrar está palavra no documento usando muito menos memória do sistema.

3.9.1. SAX

A API SAX é mais voltada para a sintaxe, enquanto DOM oferece muitas funcionalidades para desenvolver algumas aplicações com dados XML. A SAX é um API padrão para análise (*parsing*) documentos XML. Um *parser* SAX lê o fluxo de dados XML, analisa-os e detecta eventos ao interpretar as *tags*. Estes eventos são coletados pela aplicação que pode realizar ações específicas para cada um deles.

Para entender como funciona uma API baseada em eventos (como e o caso da SAX), considere o documento da Lis*tag*em 32, ao ser analisado por uma API baseada em eventos. A API irá dividir a estrutura deste documento em uma série de eventos lineares tais como : *start document*; *start element*: doc; *start element*: para; *characters*: Alo, mundo!; *end element*: para; *end element*: doc; *end document*;

```
<?xml version="1.0"?>
<doc>
   <para>Alo, Mundo!</para>
</doc>
```

Lis*tag*em 32 : Documento para ser processado através de SAX.

A aplicação por sua vez irá tratar cada um destes eventos tal como trata eventos gerados por uma interface gráfica de um usuário: não há necessidade de armazenar o documento inteiro na memória ou em um meio de armazenamento secundário.

3.9.2. DOM

O DOM é uma API para documentos HTML e XML. Ela define a estrutura lógica dos documentos e como estes documentos serão acessados e manipulados. Na especificação do DOM o termo documento (*document*) é largamente usado. XML está sendo usado como uma maneira de representar muitos tipos diferentes de informações que podem estar armazenadas nos diversos sistemas e o DOM é usado para manipular estes dados [Wood, 1998].

Com o DOM, os programadores podem construir documentos, navegar na sua estrutura, modificar ou apagar elementos no seu conteúdo. Qualquer dado presente em um documento XML pode ser acessado, alterado, excluído, ou adicionado usando o DOM. Sendo uma especificação da W3C, o objetivo mais importante do DOM é prover uma interface padrão para programação que possa ser usada por uma grande quantidade de ambientes e aplicações. O DOM foi desenvolvido para ser usado com qualquer linguagem de programação.

No DOM os documentos tem uma estrutura lógica que é muito semelhante a uma árvore. A representação de uma tabela HTML da Lis*tag*em 33, tem a estrutura lógica representada no DOM conforme a Figura 15.

```
<TABLE>
<TBODY> <TR> <TD>Shady Grove</TD>
<TD>Aeolian</TD> </TR> <TR>
```

```
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD> </TR> </TBODY>
</TABLE>
```

Lis*tag*em 33 : Representação de uma tabela em HTML.

O nome "*Document Object Model*" foi escolhido por que ele é um modelo de objetos dentro do desenvolvimento orientado a objetos : os documentos são modelados usando objetos e o modelo agrega não apenas a estrutura deste documento mas também o comportamento do documento e dos objetos que o compõe. Em outras palavras, os nós da Figura 15 não representam a estrutura dos dados, eles representam os objetos os quais possuem funções e identidade.

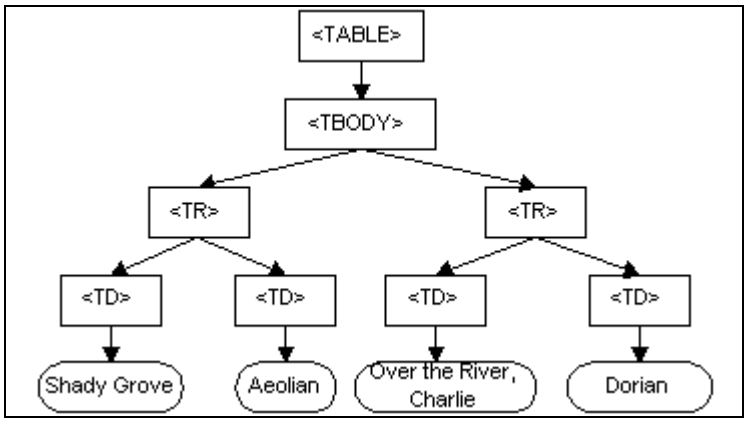

Figura 15 : Representação lógica de uma tabela HTML em um DOM.

Segundo [Wood, 1998], a estrutura de um documento SGML é tradicionalmente representada por um modelo abstrato, e não por um modelo de objetos. Em um modelo abstrato, o modelo é centrado nos dados. Nas linguagens de programação orientadas a objetos, os dados estão encapsulados nos objetos que armazenam estes dados, não sendo permitido que os mesmos sejam manipulados diretamente por objetos externos. As funções associadas a estes objetos determinam como estes objetos podem ser manipulados, e elas são parte do modelo de objetos.

O DOM atualmente está divido em duas partes : DOM Core - representa as funcionalidades utilizadas para manipular documentos XML e também serve como base para a DOM HMTL, e o DOM HTML que representa as funcionalidades utilizadas para manipular documentos HTML.

3.10. Integridade em documentos XML

As questões referentes a manutenção de restrições de integridade em documentos XML vem recebendo grande atenção pela comunidade de pesquisadores e muitos esforços tem sido feitos no sentido de se criar maneiras e padrões para a definição e validação de regras de integridade como por exemplo chaves primárias ou chaves estrangeiras.

Várias propostas foram feitas no sentido de se definir e validar restrições de integridade em documentos XML utilizando os padrões da especificação XML como através da DTD, XML Schema, entre outros [Buneman et ali, 2001].

O maior problema para se determinar restrições de integridade em um documento XML está em como especificar que, por exemplo, em um determinado nível da árvore XML, um determinado elemento será chave primária, ou seja, ele só poderá existir uma vez naquele nível, ou como, por exemplo, dizer que um atributo deste mesmo elemento será chave estrangeira de elementos que se encontram em outros níveis da árvore. Para se resolver este problema, é necessário a utilização de um padrão para se especificar os caminhos dentro da árvore XML para assim se poder acessar a um determinado nível da árvore.

Para [Buneman et al., 2001], chaves são de fundamental importância em uma base de dados, elas proporcionam um meio eficiente para se localizar objetos e relacionar uma objeto com o outro (relacionamentos), e as mesmas são de grande importância na validação de dados, permitindo mantêr os mesmos de acordo com o modelo definido baseado no mundo real. Muitas propostas para manutenção de chaves foram feitas usando o próprio padrão XML através de DTD e XML Schema para manutenção das mesmas, mas estas propostas se mostram deficientes em muitos casos. Como um documento XML não precisa necessariamente possuir uma definição de sua gramática (através de DTD, XML Schema), é muito útil que o mesmo possua um mecanismo para especificação de chaves independente do tipo de documento XML.

Para superar estas limitações, autores propõe mecanismos para definição de chaves que :

- possam ser definidos através de uma ou mais expressões de caminho, possibilitando indicar precisamente um determinado nó em uma árvore XML;

- possam ser definidas para um conjunto relativo de nós;
- não dependa de qualquer mecanismo utilizado para especificar a estrutura do documento, como por exemplo DTD e XML Schema.

[Chen et ali., 2002] propôs uma notação para definição de chaves com a seguinte sintaxe :

$$(Q, (Q', \{P_1,...,P_P\}))$$

onde Q, Q', e $P_1,...,P_P$ são expressões de caminho[12]. Q é chamado de caminho do contexto (*context path*), Q' é o caminho alvo (*target path*) e $P_1,...,P_P$ são caminhos chaves (*keys path*). A idéia é que o caminho de contexto Q identifique o conjunto de nós do contexto (*context nodes*) onde para cada nó *n* a restrição de integridade deve ser respeitada no conjunto de nós acessíveis através do caminho alvo Q'.

Por exemplo, a chave KS_1 definida por : KS_1= (/, (./*universidade*, {./*nome*})) indica que no caminho de contexto "/" (ou seja raiz do documento) uma "universidade" é identificada por um subelemento chamado "nome". Outro exemplo, tomemos uma chave KS_2= (/*universidade*, (.//*funcionario*, {./@*codigofuncionario*})) que indica que no dentro de um nível do documento identificado pelo elemento universidade (caminho de contexto), um funcionário em qualquer subnível (indicado pelo ".//") do documento dentro do caminho de contexto é identificado pelo por um atributo ("@") do elemento funcionário denominado "codigofuncionario". Como um terceiro exemplo, tomemos uma chave KS_3= (/, (.//*funcionario*, {./*nome*, /*telefone*})) que indica que no caminho de contexto "/", ou seja todo o documento, um elemento "funcionario" em qualquer subnível da árvore é identificado unicamente por um elemento denominado "nome" e um elemento denominado "telefone".

Esta notação é complexa, mas permite definição de restrições de integridade em documentos XML. Existem muitas outras propostas diferentes e com estudos aprofundados propondo algoritmos de validação para verificar um documento está de acordo com um conjunto de restrições para ele pré-determinadas.

3.11. Comentários finais

[12] Expressões de caminho (*Path Expressions*) foram abordados na seção 3.5.1.

O padrão XML é uma poderosa ferramenta capaz de representar dados que não são possíveis de se representar através dos modelos tradicionalmente utilizados, tais como os modelos relacional e objeto. Esta van*tag*em se deve pelo fato de que os dados XML são auto-descritivos, ou seja, sua estrutura esta representada juntamente com seu conteúdo através da organização e aninhamento de seus elementos, e ao contrário da HTML, não armazena informações sobre como estes dados serão apresentados. O padrão XML separa os dados das questões de apresentação, permite que os mesmos sejam formatados e apresentados – através de uma folha de estilos (XSL) por exemplo - de diversas maneiras e para os mais variados fins.

Os padrões para definição de esquemas tem um papel fundamental na definição de gramáticas para os dados XML, dentre elas, merecem destaque a DTD e a XML Schema, esta última muito mais poderosa e em constante evolução.

Linguagens de consulta para dados XML permitem são uma ferramentas poderosas que permitem a execução de consultas em dados XML possibilitando extrair e/ou alterar os mesmos de maneira muito funcional, tornado a manipulação de um documento XML tão simples como a manipulação de tabelas no modelo relacional.

A questão da integridade referencial em documentos XML é muito importante quando o padrão XML é utilizado para representar dados relacionais, sendo que os padrões para definição de esquemas relacionados a XML (DTD, XML Schema e outros) deixam a desejar quando se deseja manter em um documento, regras de integridade as quais dados exportados de bases relacionais estavam submetidos, tais como chaves primárias e estrangeiras.

4.Integração de Fontes Heterogêneas de Dados Utilizando XML

Este capítulo aborda algumas técnicas utilizadas para representar dados relacionais através de XML, e uma vez que estes dados relacionais tenham sido exportados para XML, como podemos acessá-los, mantê-los e organizá-los e integrá-los através do conceito de visões materializadas.

Também neste capítulo serão estudadas as questões ligadas a utilização de visões de dados XML para representação e integração de fontes heterogêneas de dados, bem como, algumas propostas já existentes para integrar dados de diversas fontes heterogêneas.

4.1. Representando Bases de Dados Relacionais com XML

Uma base de dados relacional é representada por seu esquema como por exemplo r1(a,b,c) r2(d,e). Nestas expressões, r1 e r2 são os nomes das relações (tabelas), a,b,c são colunas da relação r1 e d,e são colunas da relação r2.

Tomemos então, por exemplo um base relacional formada por duas relações r1 e r2 descritas dentro de uma notação {r1 : i_1, r2 : i_2} onde i_1 e i_2 representam os dados presentes nas respectivas relações que podem ser representados através de um conjunto de linhas (tuplas) conforme a Lis*tag*em 34.

```
{r1 : {tupla{a : a1, b : b1, c : c1},
      {tupla{a : a2, b : b2, c : c2},
},
{r2 : {tupla{d : d1, e : e1},
      {tupla{d : d2, e : e2},
      {tupla{d : d3, e : e3}
}
```

Lis*tag*em 34 : Representação das relações r1 e r2 através de tuplas.

As relações r1, r2 podem também ser representadas através de tabelas como podemos observá-las respectivamente na Tabela 6 e na Tabela 7.

r1:		
a	b	c
a1	b1	c1
a2	b2	c2

Tabela 6 : Exemplo de relação r1.

r2:	
d	e
d1	e1
d2	e2
d3	e3

Tabela 7 : Exemplo de relação r2.

Segundo [Abiteboul et al., 2000], podemos representar estes dados através de uma estrutura de árvore das mais variadas formas, dependendo da organização desejada. As figuras Figura 16, Figura 17 e Figura 18 mostram três maneiras de representar dados das relações r1 e r2 através de árvores.

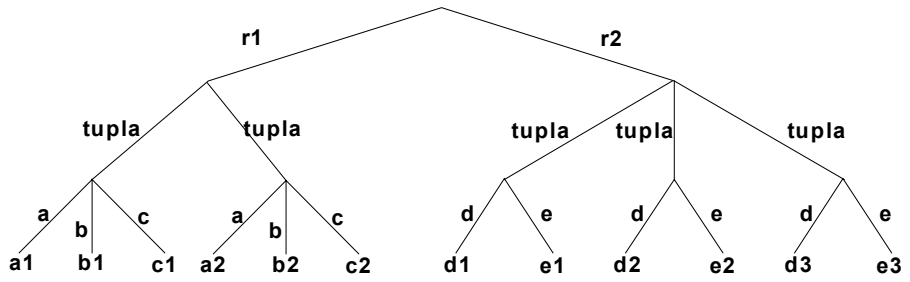

Figura 16 : Representação de dados relacionais em árvore – exemplo 01.

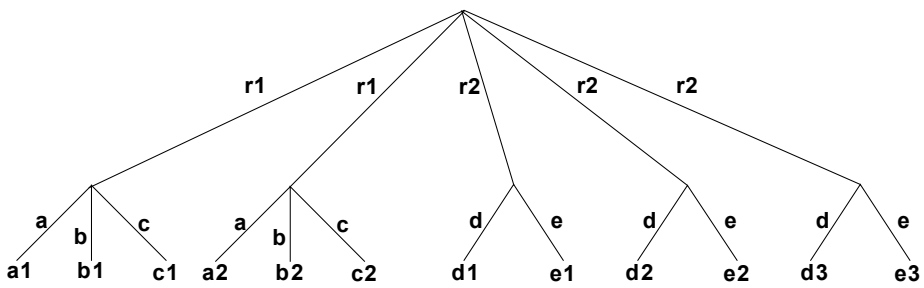

Figura 17 : Representação de dados relacionais em árvore – exemplo 02.

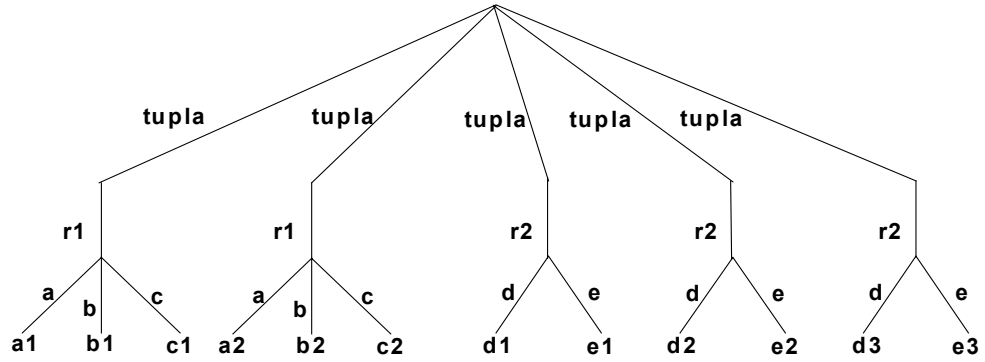

Figura 18 : Representação de dados relacionais em árvore – exemplo 03

Cada uma das árvores representadas na Figura 16, Figura 17 e Figura 18 é pode ser representada através de um documento XML. A Lis*tag*em 35 mostra como podem ser representados os dados das relações r1 e r2 em um documento XML de acordo com a representação em árvore da Figura 16.

```
<?xml version="1.0"?>
<bd>
    <r1>
        <tupla>
            <a> a1 </a>
            <b> b1 </b>
            <c> c1 </c>
        </tupla>
        <tupla>
            <a> a2 </a>
            <b> b2 </b>
            <c> c2 </c>
        </tupla>
    </r1>
    <r2>
        <tupla>
            <d> d1 </d>
            <e> e1 </e>
        </tupla>
        <tupla>
            <d> d2 </d>
            <e> e2 </e>
        </tupla>
        <tupla>
            <d> d3 </d>
            <e> e3 </e>
        </tupla>
    </r2>
</bd>
```

Lis*tag*em 35 : Representação da árvore de dados da Figura 16 com XML.

Desta maneira, podemos representar os dados de uma mesma base relacional em documentos XML com estruturas variadas. Esta estrutura esta será determinada pela

organização e pela formatação exigida por quem (usuário/aplicação/desenvolvedores) irá eventualmente utilizar estes dados.

4.2. Visões XML

Visões materializadas em banco de dados são aquelas para as quais ocorre o armazenamento físico das informações, obtidas a partir das fontes de dados originais. Consultas podem ser respondidas, na maior parte dos casos, sem o acesso direto às fontes de informação que originaram a visão.

Uma visão materializada se caracteriza pelo armazenamento de suas tuplas no banco de dados, com a construção de estruturas de índices próprias e a possibilidade de acessos mais rápidos do que aqueles aplicados sobre os dados que originaram a visão [Gupta et al., 1995]. Uma visão materializada se constitui em uma cópia ("cache") de determinados dados, apropriada para consultas que exijam respostas rápidas ou quando se queira evitar, sempre que possível, o acesso aos dados de origem. Este tipo de visão é definida como uma alternativa à abordagem virtual (por demanda) [Widom, 1995], possuindo as seguintes características:

- informações das fontes de dados que sejam identificadas como potenciais itens de consulta são extraídas, convertidas, filtradas e tratadas em conjunto com outras possíveis informações relevantes, sendo posteriormente armazenadas em um repositório centralizado de dados;
- quando uma consulta é submetida, esta é avaliada diretamente junto ao repositório criado, sem que seja necessário um acesso constante junto às fontes de dados existentes.

O uso de visões materializadas ou virtuais deve ser considerado de acordo com o tipo de situação na qual se esteja aplicando a integração de fontes heterogêneas. Neste sentido, diversas contribuições têm sido apresentadas em trabalhos desenvolvidos nos últimos anos [Hull, 1997].

A abordagem virtual tende a apresentar melhores resultados quando as fontes de informações estão mudando freqüentemente, enquanto a materialização, por seu lado, se

aplica melhor quando estas fontes não sofrem alterações com tanta freqüência, além de ser exigido um tempo de resposta rápido para as consultas submetidas [Hull, 1996].

A abordagem virtual é apropriada para informações que mudam rapidamente, para clientes com necessidades normalmente não previsíveis e para consultas que são realizadas sobre grandes quantidades de dados, a partir de um volume significativo de fontes de informações [Widom, 1995]. No entanto esta abordagem se mostra ineficiente quando consultas são submetidas repetidas vezes, quando as fontes de informação são lentas, caras ou muitas vezes não disponíveis, e quando é requerido um processamento considerável para as diversas etapas de preparação dos dados, antes de seu uso. Neste caso, o uso de materialização se torna a alternativa de melhores resultados, por permitir que a informação esteja disponível imediatamente para consultas e análises desejadas.

Em um outro artigo [Hull, 1997], Hull afirma que a materialização de visões integradas pode oferecer benefícios substanciais sobre a abordagem virtual no que se refere ao tempo de resposta para consultas, podendo isto ser observado quando são necessárias junções de grande complexidade envolvendo os dados das múltiplas bases de dados presentes na visão integrada. Além disso, também se aplica adequadamente em casos nos quais não há uma chave universal para entidades referenciadas a partir destes diversos bancos de dados.

No entanto, uma preocupação a ser considerada quando se trata deste tipo de abordagem diz respeito à atualização das visões materializadas, sendo necessário implementar algum tipo de mecanismo que permita garantir a coerência dos dados manipulados em relação aos dados armazenados nas fontes de dados integradas. Para garantir uma visão materializada consistente, algum mecanismo deve ser adotado para mantê-la em relação às fontes dos dados.

Uma visão materializada proporciona rápido acesso aos dados armazenados, podendo a diferença de velocidade ser crítica para aplicações nas quais ocorrem muitas consultas, com visões tão complexas que não se torna possível refazê-las para cada nova consulta realizada [Gupta et al., 1995]. Porém, assim como uma "cache", este tipo de visão necessita de mecanismos que permitam mantê-la adequada às mudanças ocorridas com os dados a partir dos quais esta se originou, sendo adotado normalmente um processo denominado atualização de visões.

Outra van*tag*em apresentada para justificar o uso de visões materializadas é descrita por Abiteboul [Abiteboul et al., 1998], quando este afirma que normalmente a prática de materialização é adotada para melhorar o desempenho de consultas, quando os dados de origem estão localizados remotamente, de forma distribuída, ou o tempo de resposta para consultas é um fator crítico. Da mesma forma que os demais autores, Abiteboul descreve a importância de que o conteúdo de visões desta natureza sejam sempre mantidos de forma consistente em relação às fontes de dados, seja através da sua redefinição a partir destas fontes, seja pela computação de mudanças incrementais em seus dados tendo como base as alterações que ocorrerem com as fontes originais.

4.2.1. Atualização de visões

Dados materializados, sobre os quais se realize qualquer tipo de consulta, devem ser atualizados periodicamente, buscando garantir sua coerência em relação às fontes das quais foram extraídos. O processo de atualizar uma visão materializada, em resposta à mudanças nos dados fontes, é chamado de atualização de visões [Gupta et al., 1995].

Recomputar todo o conteúdo de uma visão, a partir de mudanças nas fontes de dados, é uma alternativa possível para manter esta visão atualizada. No entanto, em muitas situações este processo pode se tornar demasiadamente oneroso, até mesmo não justificando a existência destas visões.

Uma alternativa para a solução deste problema é a utilização de algoritmos para a atualização incremental de visões, os quais basicamente permitem modificar uma parte da visão em resposta a mudanças nas fontes de dados [Gupta et al., 1995].

Uma política de atualização de visões é a decisão de quando a visão é atualizada. Uma visão pode ser atualizada dentro de uma transação que modifica as tabelas base ou a atualização pode ser adiada. O primeiro caso é denominado atualização imediata de visões e o segundo atualização adiada de visões [Gupta et al., 1995]:

- Visões imediatas: a visão é atualizada imediatamente após uma modificação na tabela base usada para criar a visão. Esta técnica permite consultas rápidas, mas aumenta o tempo das transações de atualização;
- Visões adiadas: nesta técnica, a visão é atualizada em uma transação separada, fora da transação que atualiza a tabela base usada para derivar a visão.

Diferentes políticas podem ser definidas: (1) "preguiçosa" (lazy deferred): a visão é atualizada tão tarde quanto seja possível, desde que esteja garantindo que todos os resultados das consultas submetidas estejam consistentes com os dados base. Em outras palavras, visões "preguiçosas" adiadas não precisam estar consistentes com as tabelas sobre as quais elas foram definidas, mas as consultas sobre as visões tem de ser respondidas como se a visão estivesse consistente; (2) "periódica" (periodic deferred): a visão é modificada periodicamente em intervalos preestabelecidos, em uma transação especial de atualização. Esta técnica permite consultas rápidas e não aumenta o tempo das atualizações. A desvan*tag*em é que as consultas submetidas à visão podem trazer resultados inconsistentes com os dados fonte; (3) forçada (forced delay): a visão é atualizada depois de um certo número de modificações nas tabelas base usadas para derivar a visão.

A atualização imediata de visões tem a desvan*tag*em de que cada transação de atualização implica em "*overhead*" para propagar as mudanças das tabelas base e atualizar cada visão. Este "*overhead*" cresce com o número de visões e esta abordagem não é escalonável em relação a este número de visões.

A atualização adiada de visões remove o "*overhead*" criado pela propagação e "refresh" das transações de atualização. Entretanto, impõe diferentes "overheads" nestas transações de atualização: as mudanças nas tabelas base devem ser registradas em um arquivo de "*log*", de modo que estejam disponíveis mais tarde para a operação de atualização. Esta abordagem permite que alterações de várias transações de atualização possam ser realizadas juntas, em uma única operação de propagação e "*refresh*".

Por outro lado, atualização "preguiçosa" adiada impõe um significativo "*overhead*" em transações de consulta, já que a consulta tem que esperar para que a visão materializada seja atualizada. Dependendo do tipo de aplicação, a performance da consulta pode ser melhorada usando atraso forçado ou atualização periódica. Quando a aplicação precisa de um armazenamento estável de dados, a atualização periódica tem excelente performance, como o caso de *datawarehouses* que precisam rodar longas consultas de suporte à decisão [Silva, 2000].

4.2.2. Dimensões do problema de atualização de visões

Basicamente, há quatro dimensões ao longo das quais o problema da atualização de visões pode ser estudado [Gupta et al., 1995]:

- Dimensão da informação: se refere à quantidade de informação disponível para atualização (acesso às relações e à visão materializada, conhecimento das restrições de integridade, chaves, etc);
- Dimensão de modificação: diz respeito à modificações que devem ser manipuladas pelo algoritmo de atualização da visão;
- Dimensão de linguagem: relacionada com a forma como a visão é expressa (consulta, SQL, agregação, etc)
- Dimensão de instância: refere-se às instâncias da base de dados para as quais o algoritmo será usado.

4.3. Algumas propostas existentes de integração utilizando visões materializadas

4.3.1. Sistema ARGOS

A proposta do sistema [Quan et al., 2001] é possibilitar a criação de uma visão semi-estruturada para dados *Web*, sobre as quais possam ser executadas consultas recursivas. Baseado em técnicas de atualização da linguagem XQL, o sistema propõe um ambiente distribuído com múltiplas fontes *Web*, registradas no *site* onde as visões irão residir. A arquitetura resumida do sistema pode ser representada pelo esquema da Figura 19.

Nesta arquitetura, as fontes XML são percorridas e armazenadas em estruturas DOMs persistentes. Estas estruturas ficam armazenadas na parte principal do sistema, a visão *Web* local. Esta visão, por sua vez, pode ser dividida em dois módulos:

- gerenciador de consultas: composto pelo parser de consulta, que recebe a consulta XQL e faz um parser em uma árvore de sintaxe abstrata;
- processador de consultas, que executa chamadas para a efetivação das consultas.

Gerenciador APIX: dividido em alguns módulos específicos. O inicializador APIX faz a inicialização da estrutura APIX (*Aggregate Path IndeX*) para uma determinada consulta e também a geração da visão materializada de acordo com o resultado do processamento. O atualizador APIX, com a ajuda da estrutura APIX, realiza a atualização das visões a partir de notificações vindas do detector de atualizações nas fontes *Web*.

A estratégia de manutenção do sistema ARGOS se baseia em um algoritmo que faz a atualização incremental da visão *Web*, a partir de modificações que ocorram nas fontes de dados. Esta técnica implica em custos reduzidos se comparados àqueles envolvidos na criação inicial da visão.

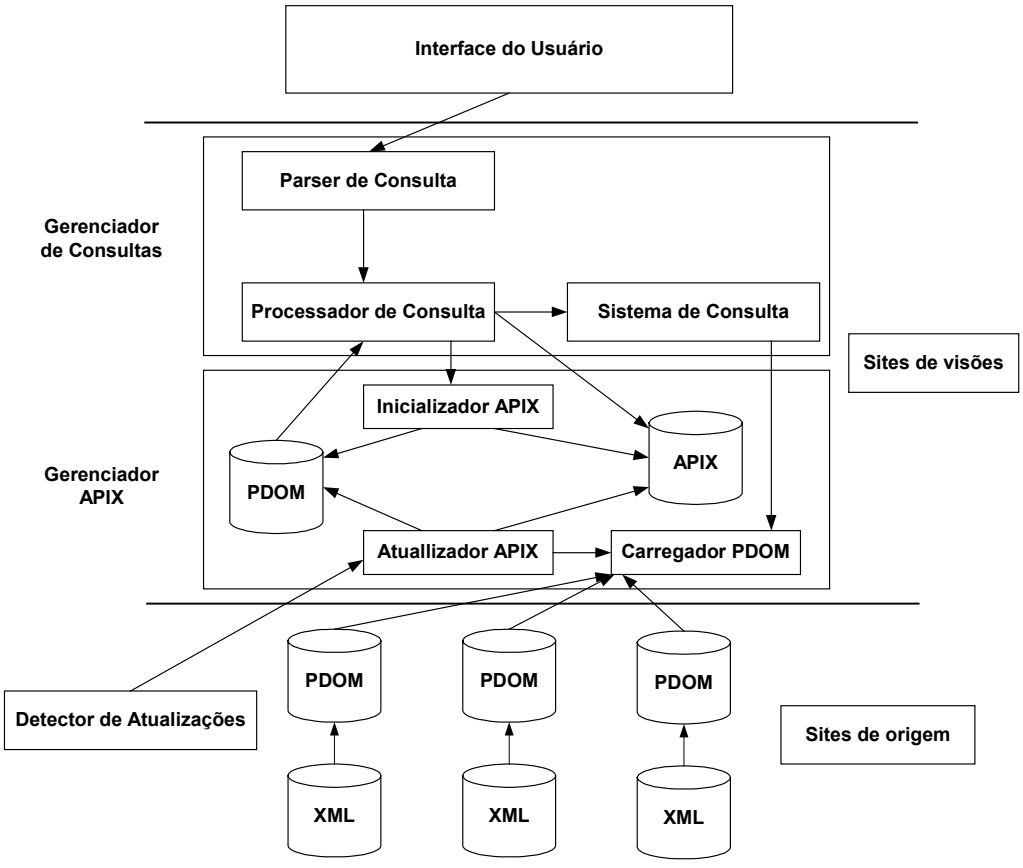

Figura 19 : Arquitetura geral do Sistema ARGOS

4.3.2. MIX (Metadata based Integration model for data X-change)

A proposta do modelo de dados MIX [Zhu et al., 2000] é oferecer uma estrutura que possibilite a criação de *datawarehouses* para dados extraídos da *Web*. O sistema

propõe um mapeamento objeto-relacional que leve em consideração as peculiaridades de *datawarehouses*, através de uma linguagem que descreva as transformações necessárias para este mapeamento.

O modelo MIX foi proposto visando atender a três desafios existentes quando da materialização de dados *Web* [Zhu et al., 2000]:

- extração de dados: dificultada pela dinamicidade e autonomia dos dados *Web*, os quais são gerados por diferentes fontes, sem um controle pré-determinado;
- preparação e integração de dados: informações semelhantes ou relacionadas, presentes na *Web*, podem ter diferentes representações de acordo com a forma como foram criadas;
- apresentação e materialização: o paradigma *Web* é totalmente diferente do paradigma de *datawarehouses*.

A idéia principal da abordagem do modelo MIX é representar os dados associados com uma descrição de seu contexto original, utilizando ontologias para interpretar corretamente estes dados e também os metadados. A arquitetura proposta, aplicando o modelo MIX, se baseia na abordagem de mediadores e está apresentada na Figura 20.

Nesta arquitetura, os *wrappers* são usados para extrair dados relevantes a partir das fontes, mapeá-los para MIX baseados em um contexto estrutural comum e retornar objetos MIX para o gerenciador de dados.

O gerenciador de dados gerencia as fontes disponíveis, através da atualização do repositório de metadados. Baseado na descrição do contexto das fontes de dados, este gerenciador tenta integrar dados heterogêneos através da conversão destes dados para um modelo semântico comum.

O servidor de ontologias armazena e gerencia o vocabulário de um domínio específico, possibilitando uma maneira de descrever conceitos da ontologia de forma independente de qualquer aplicação ou fonte de dados.

Dados integrados são repassados para o processador de transformações, o qual mapeia objetos MIX para o esquema do *datawarehouse* e carrega dados nas tabelas. Este processador se utiliza de regras de mapeamento, definidas no arquivo de

mapeamentos (o sistema propõe uma linguagem específica para a definição destas regras).

Por fim, o processador de atualização incremental modifica o *datawarehouse*, quando as fontes *Web* sofrem alterações. Este processador mantém uma cópia dos últimos objetos MIX utilizados, fazendo comparações com novos objetos MIX que sejam recebidos do gerenciador de dados.

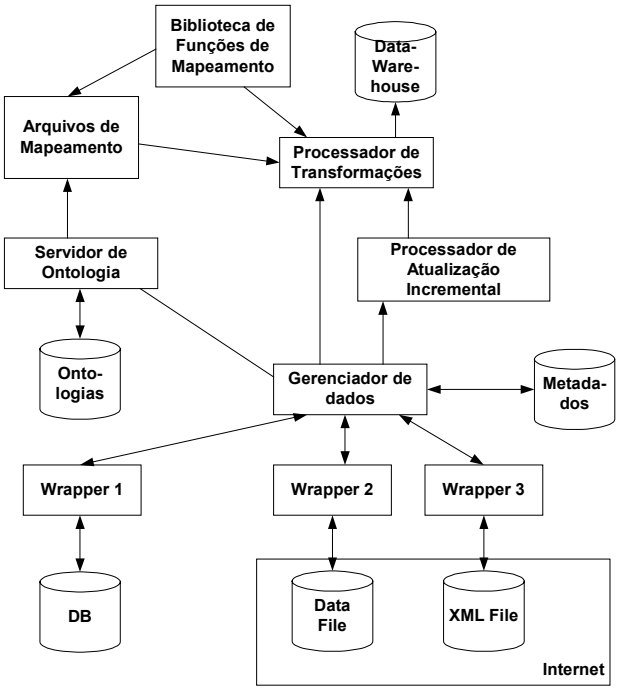

Figura 20 : Arquitetura MIX.

Um exemplo de um objeto MIX está descrito a seguir Lis*tag*em 36 : Exemplo de objeto MIX.

```
Obj = <Artigo, {
           <título, XML>,
           <Autor, {
            <nome, Ferrandin, Mauri,
               {<formato nome, Last, First>}>,
                     <endereço, {
                        <Instituição, UFSC>
}>
            }>
```

Lis*tag*em 36 : Exemplo de objeto MIX.

4.3.3. SilkRoute

A ferramenta SilkRoute [Fernandez et al., 1999] procura oferecer uma alternativa geral, dinâmica e eficiente para a criação de visões e consultas sobre dados relacionais através de XML. Este processo é realizado através de duas etapas principais:

- primeiro, é definida uma visão XML para o banco de dados relacional, através do uso de uma linguagem de consulta específica, denominada RXL (*Relational to XML Transformation Language*). Esta visão é virtual e não materializada;
- segundo passo é a formulação de consultas sobre estas visões virtuais, extraindo dados XML. Estas consultas são realizadas com uma linguagem de consulta própria para dados XML, chamada XML-QL e apenas seu resultado é enviado como resposta para a aplicação.

A arquitetura geral da ferramenta SilkRoute é ilustrada pela Figura 21.

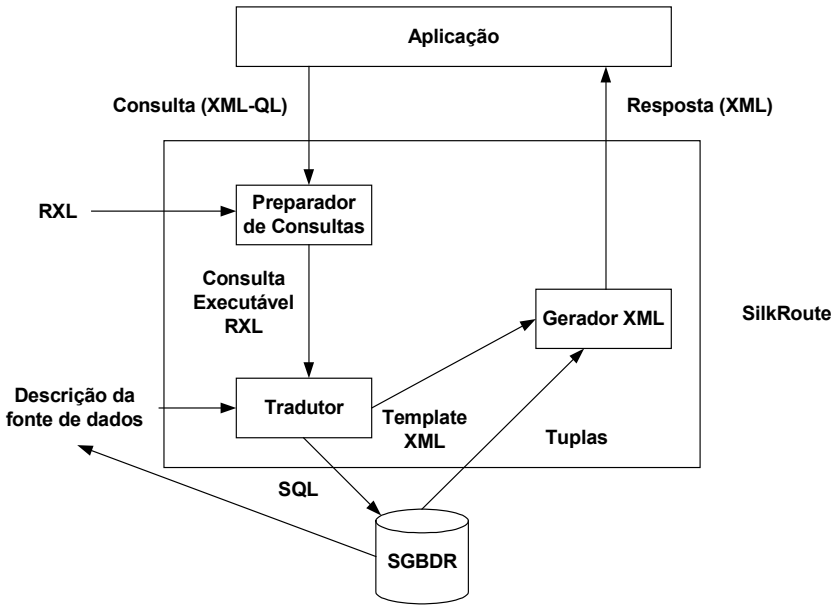

Figura 21 : Arquitetura geral do Sistema SilkRoute

A idéia principal desta arquitetura é servir como um middleware entre uma base de dados relacional e uma aplicação que acesse esta base através da *Web*. O módulo principal da ferramenta é o PREPARADOR DE CONSULTAS. A partir de uma visão virtual definida pelo administrador do banco de dados, este módulo recebe a consulta do usuário (XML-QL) e gera uma nova consulta RXL (executável) que é então enviada para o módulo TRADUTOR.

O módulo TRADUTOR, por sua vez, particiona esta consulta executável em uma ou mais consultas SQL, além de gerar dados XML (*template*) enviados ao GERADOR XML. O TRADUTOR ainda recebe como entrada uma descrição da fonte de dados, no formato XML, para auxiliar no processo de consulta de dados no banco relacional.

O módulo GERADOR XML recebe o resultado de consultas SQL e o template XML, preparando o documento XML a ser enviado como resposta para a aplicação que formulou a consulta inicial.

A linguagem RXL, que serve de base para a geração das visões nesta abordagem, combina a parte de recuperação de dados do SQL (cláusulas FROM and WHERE) com cláusulas de construção da linguagem XML-QL, gerando instruções de consulta como a apresentada na Lis*tag*em 37.

```
from ARTIGO $A
            where $A.título = "XML"
            construct   <artigo>
                                <título> $A.título </título>
                        </artigo>
```

Lis*tag*em 37 : Instruções de consulta RXL.

4.3.4. Resumo comparativo entre as principais abordagens

A tabela a seguir ilustra as principais características de cada uma das abordagens descritas segundo [Silva, 2000].

Abordagem	VISÃO MATERIALI-ZADA	OBJETIVO	ATUALIZAÇÃO	LINGUAGEM DE CONSULTA
ARGOS	Sim	Sistema de "caching" para *Web*	Incremental	XQL
MIX	Sim	Materializar dados *Web* para OLAP e DSS (*datawarehouse*)	Incremental	Qualquer (extração de dados)
SILKROUTE	Não	Middleware entre banco de dados relacional e aplicação acessando dados via *Web*	(Não se aplica)	XML-QL

Tabela 8 : Comparativo entre abordagens de visões sobre dados semi-estruturados

4.4. Comentários finais

O padrão XML é uma poderosa ferramenta para integração de dados, sendo que pode ser utilizado também para representar exportados de bases relacionais, uma vez que dados relacionais podem ser representados através de uma estrutura de árvore, que

os mesmos dados extraídos de um modelo tradicional podem ser estruturados de maneiras diferentes de acordo com a utilização que será dada aos mesmos.

Outra questão chave para integração de dados através do padrão XML é a utilização de visões XML, que servem para representar um conjunto de dados exportados ou extraídos de uma ou mais fontes de dados. Estas visões podem ser materializadas – quando ocorre armazenamento físico dos dados exportados - ou virtuais – quando não ocorre armazenamento físico e sim apenas lógico dos dados exportados. Existem diversos fatores que devem ser levados em conta no momento de definir que tipo de visão utilizar, como por exemplo, volume de dados, tipo de acesso (consulta/atualização), quantidade de acessos que cada visão irá receber, entre outros.

Os modelos apresentados (ARGOS, MIX e SilkRoute) são capazes de integrar informações das mais variadas fontes, mas nenhum trata de questões específicas presentes em dados exportados de bases relacionais, tais como a integridade referencial e a utilização do XML como um padrão único dentro do modelo, tanto para exportar os dados como regras de integridade e dicionário de dados.

5.Modelo para Integração de Fontes Heterogêneas de Dados

Este capítulo trata da proposta de um sistema para integrar dados de bases relacionais heterogêneas e distribuídas, apresentando uma proposta global que envolve todas os processos necessários para implementação de uma arquitetura mediadora para possibilitar tal integração.

Muitas questões precisam ser levadas em conta quando se propõe integração de dados de fontes relacionais heterogêneas, sendo que está proposta esta focada na integração dos dados de diversas bases relacionais através da criação de visões XML, permitindo que estes dados integrados possam ser consultados de maneira integrada e transparente para as aplicações e especificando um mecanismo para manutenção da integridade referencial a qual os dados estavam submetidos nas bases antes de serem exportados, a fim de garantir a serialização entre a visão integrada e as bases, sem violações de integridade.

5.1. Visão geral do sistema proposto

O sistema proposto consistirá de um conjunto de módulos de software que terão por objetivo viabilizar a integração de diversas fontes de dados relacionais heterogêneas e distribuídas utilizando o padrão XML para o intercâmbio dos dados entre as mesmas com o propósito de prover estes dados de maneira transparente para aplicações e desenvolvedores. Para que tal integração seja possível, será especificada uma arquitetura baseada em mediadores conforme demonstra a Figura 22.

Um conjunto de *wrappers* específicos para cada fonte de dados farão a exportação dos dados relacionais para o padrão XML, criando uma visão XML local para cada fonte a ser integrada. Estas visões serão depois integradas compondo uma visão XML global, possibilitando assim que os dados armazenados nas diversas bases heterogêneas possam ser manipulados em um única visão através de uma linguagem para consulta a dados XML.

A Figura 22 demonstra a arquitetura geral do sistema proposto, sendo que cada módulo do sistema será detalhado na seção 5.2.2.

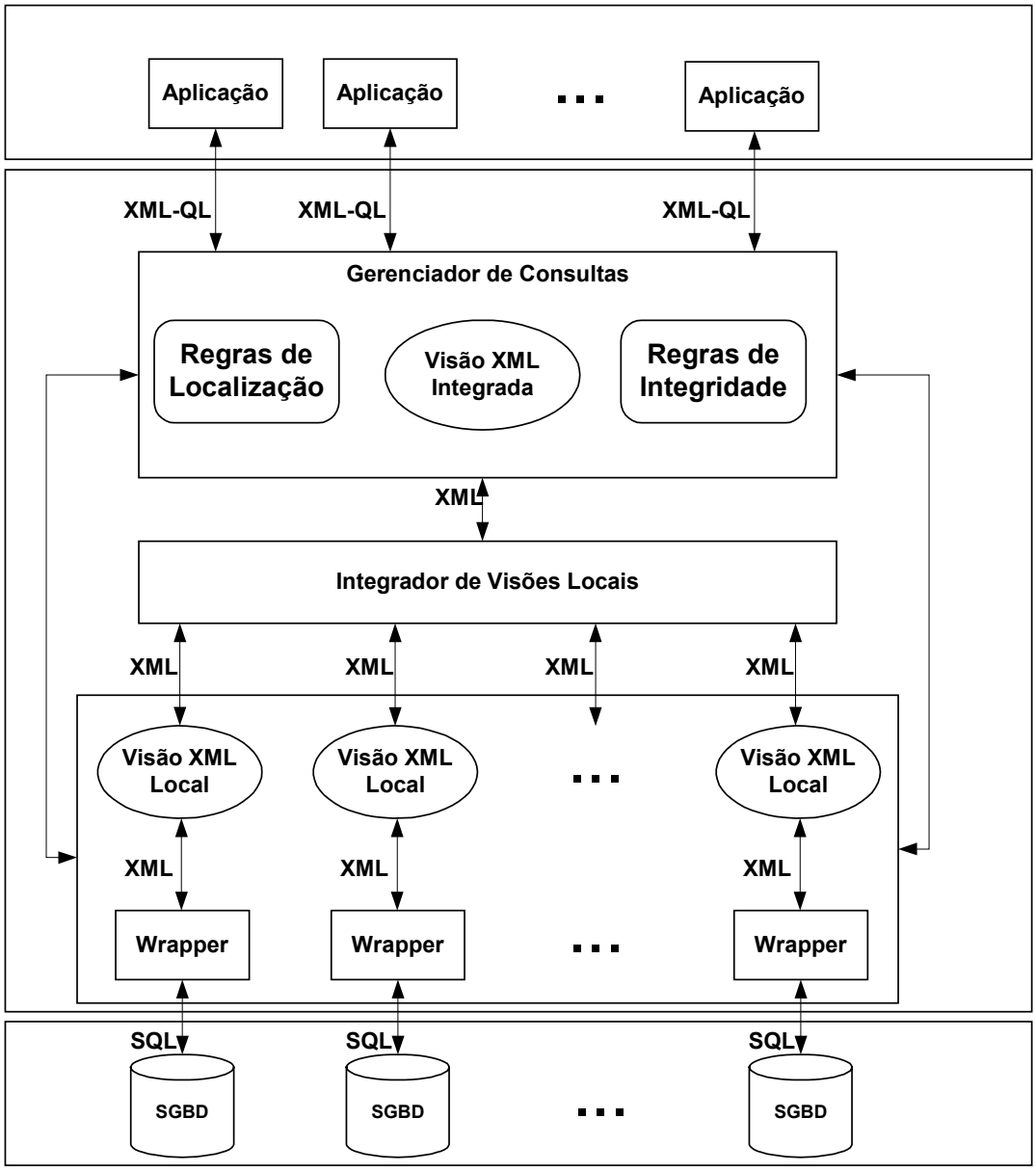

Figura 22 : Modelo proposto para integrar fontes heterogêneas de dados.

Outro ponto importante desta proposta é a manutenção das regras de integridade na visão XML global de modo que as alterações nela realizadas, possam ser propagadas para as bases relacionais sem que ocorram problemas de integridade referencial.

Para gerenciamento das regras de integridade que determinarão a integridade da visão XML global será criado um repositório de regras de integridade que armazenará as mesmas também através do padrão XML. Para definição das regras, será necessário a

intervenção de um especialista humano que definirá a partir dos dados que serão exportados, que integridade deverá ser preservada na visão XML global. O especialista deverá levar em conta a manutenção de uma integridade mínima para que não ocorram problemas de integridade em casos de sincronização de dados entre a visão XML integrada e os SGBDHs.

5.2. Desenvolvimento do protótipo

Para o desenvolvimento do protótipo será utilizada a metodologia de desenvolvimento em cascata que é composta pelas etapas de estudo de viabilidade, definição dos requisitos, projeto, implementação, integração e testes, implantação e manutenção, sendo que nesta dissertação serão cumpridas apenas a segunda, terceira e quarta etapas.

5.2.1. Definição dos Requisitos

Requisitos principais do sistema :

- Integrar fontes heterogêneas de dados armazenados em bases relacionais através do padrão XML de maneira que os dados possam ser acessados de maneira transparente para o usuário.
- Prover um mecanismo de manutenção de integridade dos dados integrados, de maneira que os mesmos possam ser sincronizados novamente para as bases relacionais sem causar violações de integridade.

5.2.2. Projeto

O sistema proposto é composto por vários módulos independentes que interagem entre si que serão detalhados nas subseções seguintes, são eles:

- os *wrappers*;
- integrador de visões XML locais;
- gerenciador de consultas;
- repositório de regras de integridade;

- repositório regras de localização dos dados.

5.2.2.1. *Wrappers*

Um *wrapper* ou tradutor, é um componente de software que converte dados e consultas (*query*) de um modelo para outro [Papakonstantinou et al., 1995]. Neste caso, uma aplicação (que pode ser um mediador), solicita ao *wrapper* consultas em uma linguagem de consulta comum (SQL, XML-QL), e o mesmo converte esta consulta para uma linguagem de consulta suportada pelo banco de dados ao qual está ligado e envia uma requisição ao mesmo, depois, recebe o resultado desta consulta e o converte para um formato suportado pela aplicação ou mediador.

No caso específico deste sistema, os *wrappers* farão acesso as informações armazenadas nas bases relacionais heterogêneas distribuídas através da SQL e converterão os resultados para o padrão XML.

Vejamos um exemplo básico do funcionamento dos *wrappers* :

Dadas as relações Paciente e Internação, representadas respectivamente na Tabela 9 e Tabela 10, presentes em uma base de dados base01:

Paciente :

CPF	Nome
01	João
02	Zé

Tabela 9 : Exemplo de tabela de pacientes base01.

Internação :

CPF	Doença
01	Gripe
01	Diarréia
02	Sífilis

Tabela 10 : Exemplo de tabela de internação base01.

Como resultado teríamos a visão XML representada pela Lis*ta*gem 38 para a base de dados base01:

```
<base01>
    <tabelas>
        <pacientes>
            <linha>
                <CPF>1</CPF>
                <Nome>João</Nome>
            </linha>
```

```
            <linha>
                <CPF>2</CPF>
                <Nome>Zé</Nome>
            </linha>
        </paciente>
        <Internação>
            <linha>
                <CPF>1</CPF>
                <Doença>Gripe</Doença>
            </linha>
            <linha>
                <CPF>1</CPF>
                <Doença>Diarréia</Doença>
            </linha>
            <linha>
                <CPF>2</CPF>
                <Doença>Sífilis</Doença>
            </linha>
        </Internação>
    </tabelas>
</base01>
```

Lis*tag*em 38: Visão XML gerada por *wrapper* a partir de uma base relaciona base01.

A implementação dos *wrappers* não é o foco principal deste trabalho, sendo que para desempenhar as funções deste módulo será utilizada uma implementação já existente chamada DB2XML que é um projeto de Volker Turau Copyright (C) 1999 distribuído sob a GNU (*General Public License*) de acordo com as normas da *Free Software Foundation*.

O DB2XML foi implementado em Java e utiliza o JDBC (*Java Database Connectivity*) para fazer acesso a bases relacionais, assim, não existe a necessidade de implementação de um *wrapper* para cada tipo diferente de SGBD, uma vez que basta carregar o driver JDBC específico para cada SGBD pois está API é dividida em duas sub APIs que são a API da linguagem e a API do fabricante do SGBD.

A Figura 23 detalha as partes da API JDBC e a Figura 24 mostra como o DB2XML será utilizado dentro do modelo proposto.

A maioria dos fabricantes de SGBD possuem API JDBC para seus produtos, sendo que a lista de todos os SGBDs que já possuem suporte pode ser encontrada no *site* http://industry.java.sun.com/products/jdbc/drivers mantido pela Sun Microsystems.

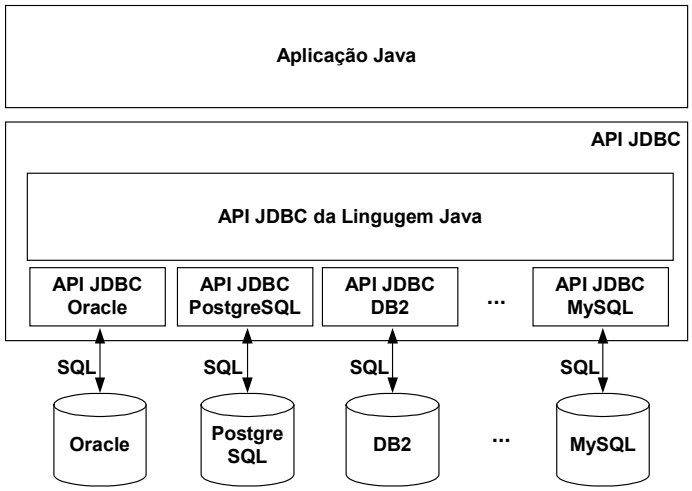

Figura 23 : API JDBC.

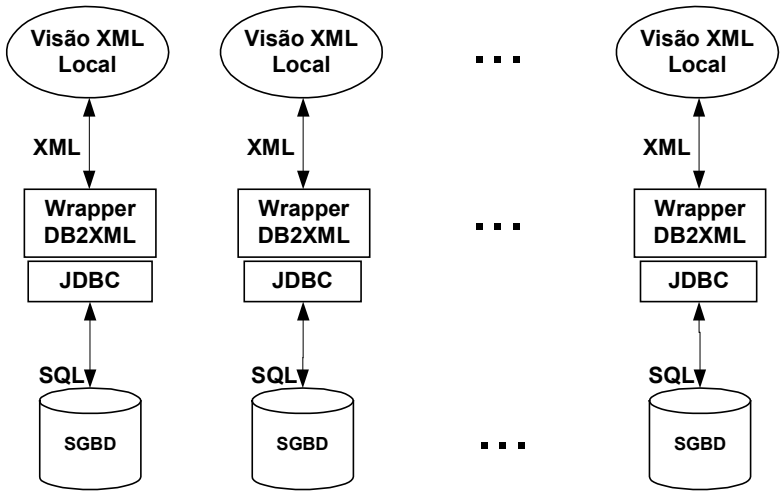

Figura 24 : Modelo proposto utilizando o *wrapper* DB2XML

5.2.2.2. Integrador de visões XML

Este módulo irá receber os dados de cada uma das visões XML locais exportadas através dos *wrappers* e os integrará em uma única visão XML global. Tal integração será feita de maneira simplificada criando um elemento raiz dentro do qual estarão contidas as visões locais. É muito importante a presença de um especialista humano no momento de inclusão/exclusão de SGBDs componentes do sistema para resolução de conflitos e problemas de integridade que eventualmente possam surgir.

A Figura 25 mostra como ficarão organizados os dados de cada visão XML local de maneira a compor uma visão XML integrada. Na raiz do documento XML que

corresponde a visão integrada foi chamado de <visao_xml_integrada>, e cada uma das bases a serem representadas visão integrada é identificada por um elemento raiz <basexx> que será adicionado ao documento como sendo elemento filho do elemento raiz.

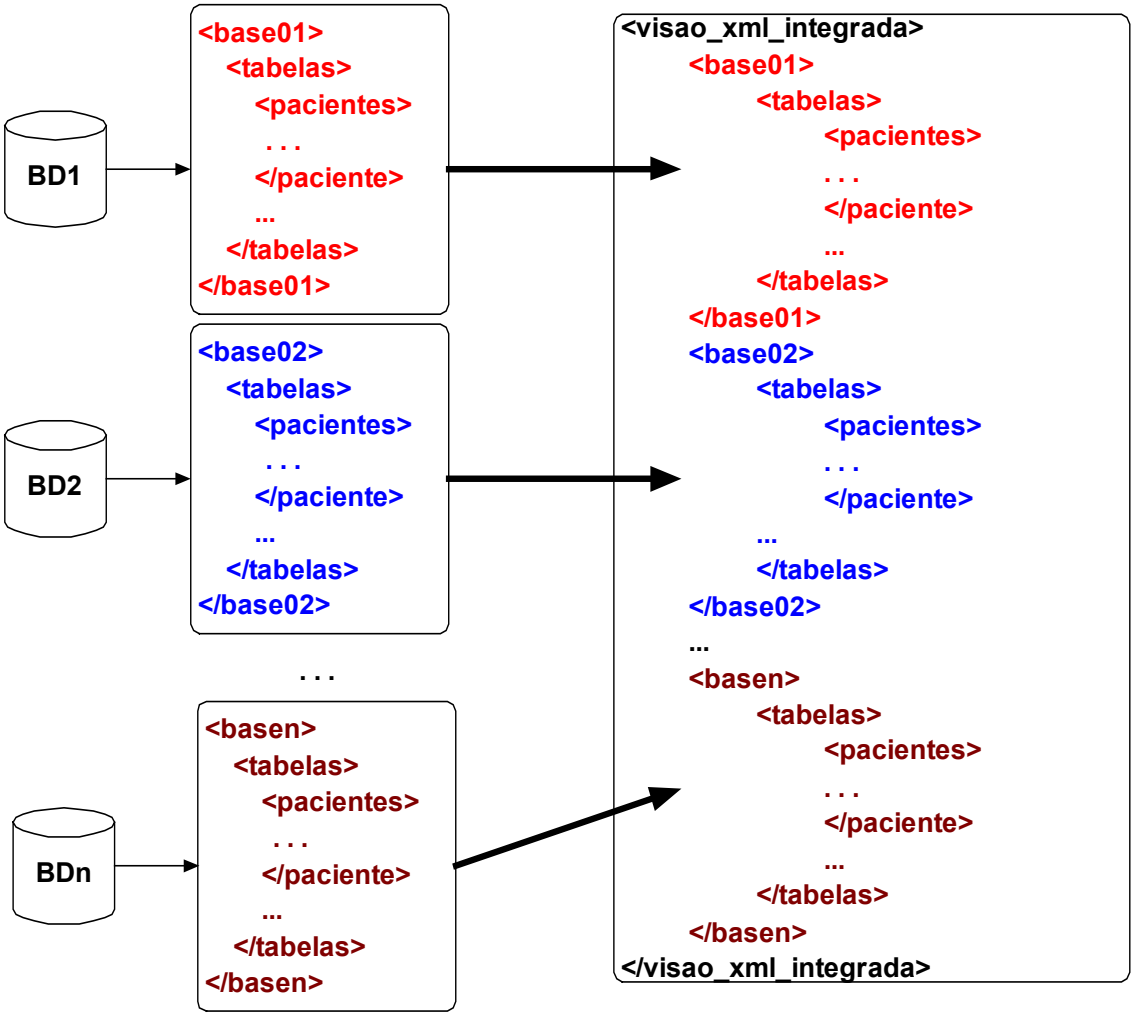

Figura 25 : Integração das visões XML locais em uma visão XML integrada

5.2.2.3. Repositório de regras de integridade

As regras de integridade da visão XML global são definidas a partir das regras de integridade de cada base relacional exportada. Para o armazenamento destas regras também será utilizado uma notação baseada em XML o que padroniza a troca e consulta de informações entre os módulos do sistema, uma vez que assim, as regras de integridade ficam acessíveis a qualquer linguagem de consulta para dados XML.

A Lis*tag*em 39 mostra o exemplo de um arquivo de regras de integridade para um documento XML conforme o proposto neste trabalho.

```
<?xml version="1.0" encoding="UTF-8"?>
<RegrasIntegridade>
   <visao_xml_integrada.basededados01.paciente.paciente_rectipo="pk">
       <campo>cpf</campo>
   </visao_xml_integrada.basededados01.paciente.paciente_rec>

    . . .
</RegrasIntegridade>
```

Lis*tag*em 39 : Exemplo de um arquivo de regras de integridade para um documento XML.

Uma regra é composta basicamente por uma *tag* que indica o contexto(path) do documento onde a regra deve ser observada, sendo que o tipo da regra é indicado por um atributo denominado "tipo" que pode ser :

- pk : para indicar que a regra é uma regra de chave primária (*primary key*);
- fk : para indicar que a regra é uma regra de chave estrangeira (*foreign key*).

A *tag* que discrimina uma regra terá então uma ou mais *tags* filhas denominadas "campos" que irão armazenar quais as *tags* que não devem possuir o mesmo valor dentro daquele contexto.

Basicamente a regra acima descrita é uma regra do de chave primária (tipo="pk") e indica que no nível do documento XML indicado pelo caminho "visao_xml_integrada.basededados01.paciente.paciente_rec" não poderão existir duas *tags* filhas da *tag* "paciente_rec" contendo o mesmo valor.

A Figura 26 representa graficamente uma violação de chave primária em um documento XML e a Lis*tag*em 40, mostra um exemplo de regra de chave estrangeira para um documento XML.

Uma regra de chave estrangeira (tipo = "fk") na prática, indica que um valor só poderá existir em um local do documento (contexto) se já existir um determinado valor em outro local (contexto) do documento. Assim as regras de chave estrangeira são compostas por uma *tag* cujo o nome da mesma determina o contexto onde ela será validada, no caso do exemplo acima, o caminho é "visao_xml_integrada.basededados01.internacao.internacao_rec" com um atributo denominado "contexto" (visao_xml_integrada.basededados01.paciente.paciente_rec no caso) que indica onde se encontram os dados de referência. Na prática, de acordo com

esta regra, somente será possível inserir uma *tag* com nome “cpf” (<campo>) no contexto “visao_xml_integrada.basededados01.internacao.internacao_rec” da visão se existir uma *tag* chamada “cpf” (<campofk>) no contexto “visao_xml_integrada.basededados01.paciente.paciente_rec”.

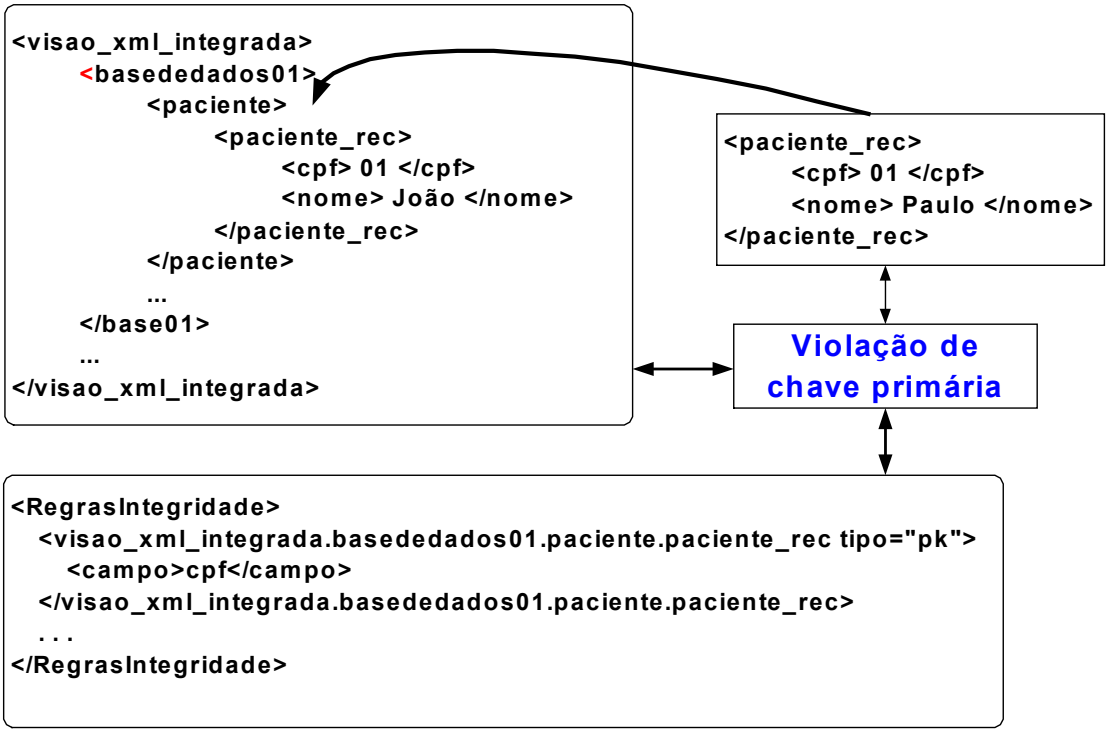

Figura 26 : Violação de chave primária na visão integrada XML.

```
<visao_xml_integrada.basededados01.internacao.internacao_rec tipo="fk"
contexto="visao_xml_integrada.basededados01.paciente.paciente_rec">
        <campo>cpf</campo>
        <campofk>cpf</campofk>
</visao_xml_integrada.basededados01.internacao.internacao_rec>
```

Lis*tag*em 40 : Exemplo de regra de chave estrangeira para um documento XML.

A Figura 27 ilustra a inserção de um valor na visão integrada considerando regras de chave estrangeira.

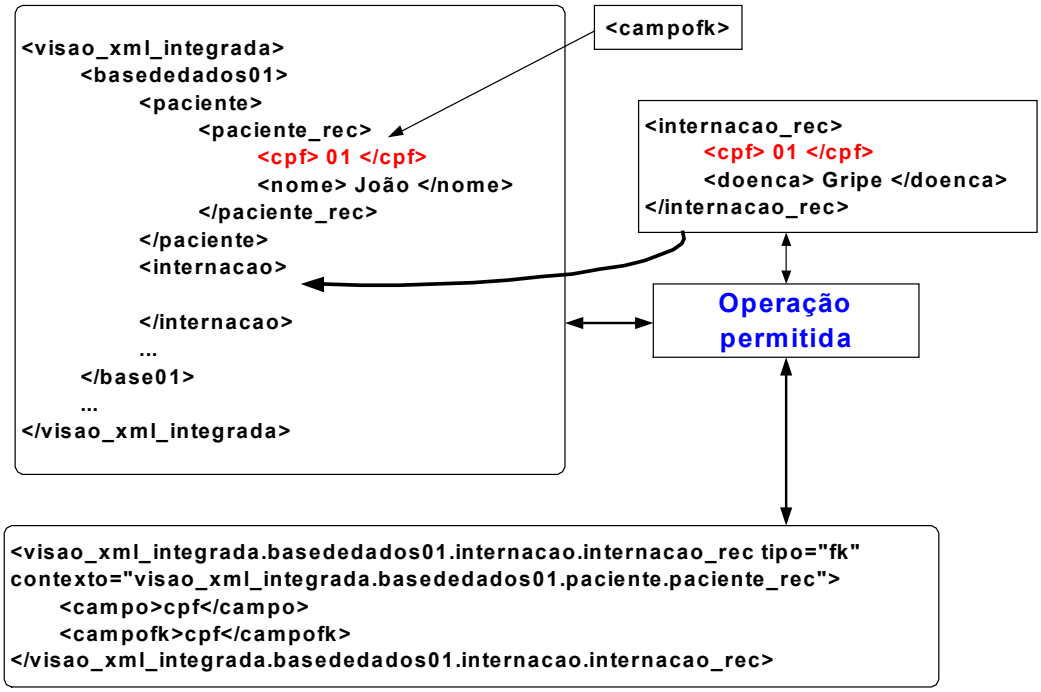

Figura 27 : Inserção de um registro na visão integrada de acordo com as regras de chave estrangeira.

5.2.2.4. Repositório regras de localização dos dados.

O repositório de regras de localização armazenará metadados referentes a origem dos dados presentes na visão XML integrada dos mesmos, tal como, qual a base de dados da qual tal campo foi exportado, tabela, etc.

A Lis*tag*em 41 mostra um exemplo de regras de localização.

```
<localizacao>
    <base nome=base01>
        <SGBD>Oracle</SGBD>
        <IP>10.0.0.1</IP>
        <porta>1521</porta>
        <campos>
            <campo>
                <nome>CPF</nome>
                <tabela>Paciente</tabela>
            </campo>
            <campo>
                <nome>Nome</nome>
                <tabela>Paciente</tabela>
            </campo>
            ...
        </campos>
    </base>
</localizacao>
```

Lis*tag*em 41 : Exemplo de repositório de dados de localização.

O repositório de regras de localização serve para facilitar uma possível propagação de alterações nos dados presentes na visão XML integrada para as bases distribuídas. Este processo é bastante complexo e não será foco de estudo deste trabalho, assim, este repositório de dados somente será usado se o gerenciador de consultas eventualmente por algum motivo precisar conhecer a origem dos dados, origem esta que deve ser escondida do usuário/desenvolvedor para manter a transparência do sistema.

5.2.2.5. Gerenciador de consultas

O módulo gerenciador de consultas tem como funcionalidade principal, receber as consultas formuladas pelas aplicações já em XML-QL e executar as mesmas sobre a visão XML global para recuperar os dados solicitados, ou sobre o repositório de regras de integridade se houver alguma atualização ou em algum caso em que o próprio gerenciador ou a aplicação neces*site*m de dados referentes a integridade da visão, ou ainda sobre o repositório de dados de localização original dos dados nas bases relacionais exportadas. A Figura 28 detalha o funcionamento do gerenciador de consultas, sendo que podemos definir como seqüência de eventos que podem ocorrer com o gerenciador de consultas através do algoritmo mostrado na Lis*ta*gem 42.

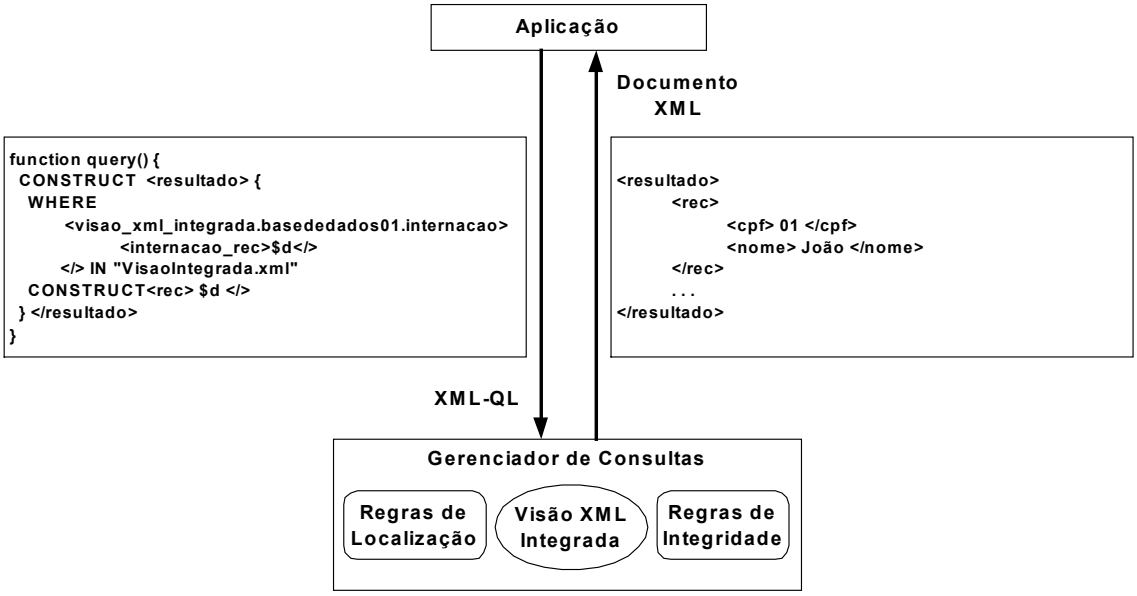

Figura 28 : Funcionamento do gerenciador de consultas.

Para a execução das consultas XML-QL sobre a visão integrada XML foi utilizado a implementação de um protótipo de processador de consultas para esta linguagem desenvolvido pela AT&T Corporation sendo que o mesmo está disponível em http://www.research.att.com/sw/tools/xmlql/.

```
...
consulta = rede.recebe_requisição_da_aplicação();
Se consulta.tipo != alteração_de_dados então
   rede.retorna_aplicação(
         gerenciador_consultas.executa_consulta(consulta)
         );
senão
   se gerenciador_consultas.verifica_integridade(
                  consulta) = OK então
         rede.retorna_aplicação(
         gerenciador_consultas.executa_consulta(consulta)
         );
   senão
         rede.retorna_aplicação(
                  gerenciador_consultas.codigo_erro()
         );
   fim se;
fim se;

    ...
```

Lis*tag*em 42 : Algoritmo básico representando o funcionamento do gerenciador de consultas.

5.2.3. Especificações através de UML

A UML (*Unified Modelating Language*) é uma linguagem para modelagem orientada a objetos. A mesma é composta de vários tipos de diagramas utilizados na modelagem, sendo que para especificação deste protótipo a relevância maior está no :

- diagrama de classes : descreve os tipos de objetos no sistema e os vários tipos de relacionamento estático que existem entre eles;
- diagrama de casos de uso : descreve um conjunto de cenários amarrados por um objetivo comum de um usuário, sendo que um cenário é uma seqüência de passos que descrevem uma interação entre o usuário e o sistema.

5.2.3.1. Diagrama de classes

O diagrama de classes que serão implementadas para desenvolvimento do sistema é demonstrado através da Figura 29, sendo que o mesmo especifica apenas as classes fundamentais para o funcionamento do protótipo.

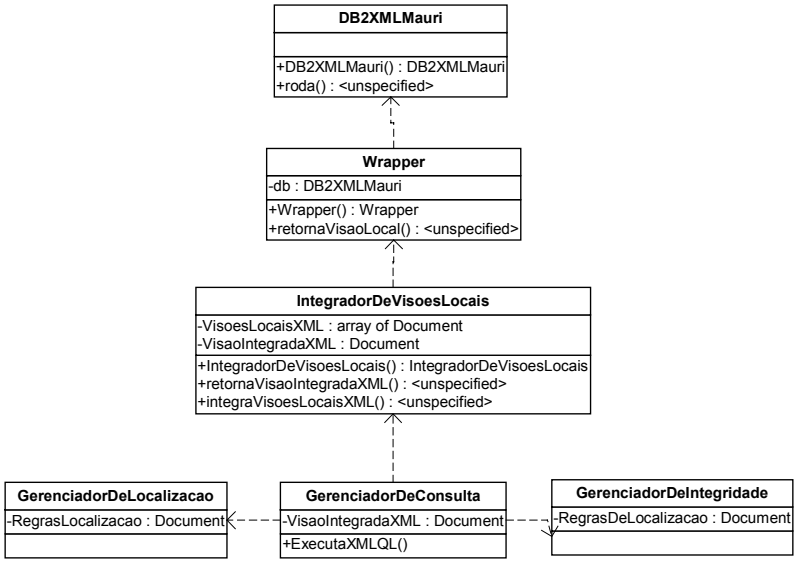

Figura 29 : Diagrama de classes do protótipo proposto.

5.2.3.2. Diagrama de casos de uso

A Figura 30 mostra o diagrama de casos de uso do protótipo proposto por este trabalho.

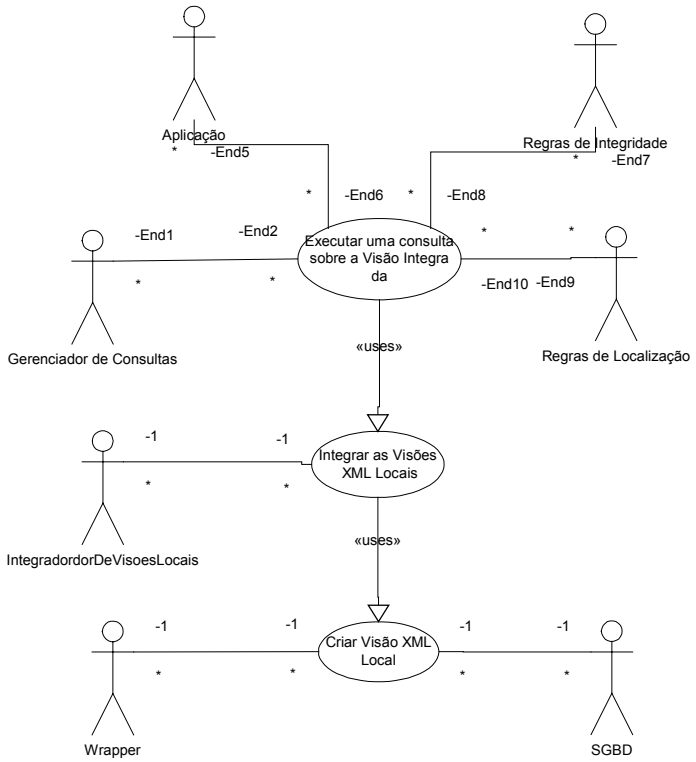

Figura 30 : Diagrama de casos de uso do protótipo proposto.

5.2.4. Implementação

Para a implementação do sistema proposto serão utilizado basicamente as seguintes ferramentas:

- Bases de dados de diferentes fabricantes - Oracle, PostgreSQL;
- Linguagem de programação Java;
- Nebeans – IDE para linguagem Java;
- JDBC para acesso as bases relacionais com Java;
- B2XML - uma implementação de um *wrapper* para bancos de dados que possibilitam acesso via JDBC;
- XML-QL - uma implementação de um módulo para execução de consultas sobre dados XML.

5.2.4.1. Implementação das classes do sistema

As classes básicas do sistema foram implementadas de acordo com a especificação UML mostrada na Figura 29, sendo que todas as classes foram implementadas utilizando a linguagem de programação Java.

Para uma melhor demonstração da implementação realizada, foi necessário a criação de uma interface gráfica simples que pudesse concentrar as operações principais do protótipo em uma única tela, de maneira a facilitar a utilização do mesmo, bem como a compreensão do sistema proposto como um todo.

5.2.5. Exemplo de utilização do protótipo

Nesta subseção será demonstrado através de figuras a utilização do protótipo na realização de algumas operações básicas.

A Figura 31 mostra a interface inicial do protótipo, composta basicamente de um conjunto de botões que serão utilizados para disparar diversos eventos específicos, tais como :

- materializar as visões : dispara o processo de extração dos dados relacionais das bases para XML através dos *wrappers*, e na seqüência o processo de integração das visões criando assim a visão XML integrada dos dados relacionais;

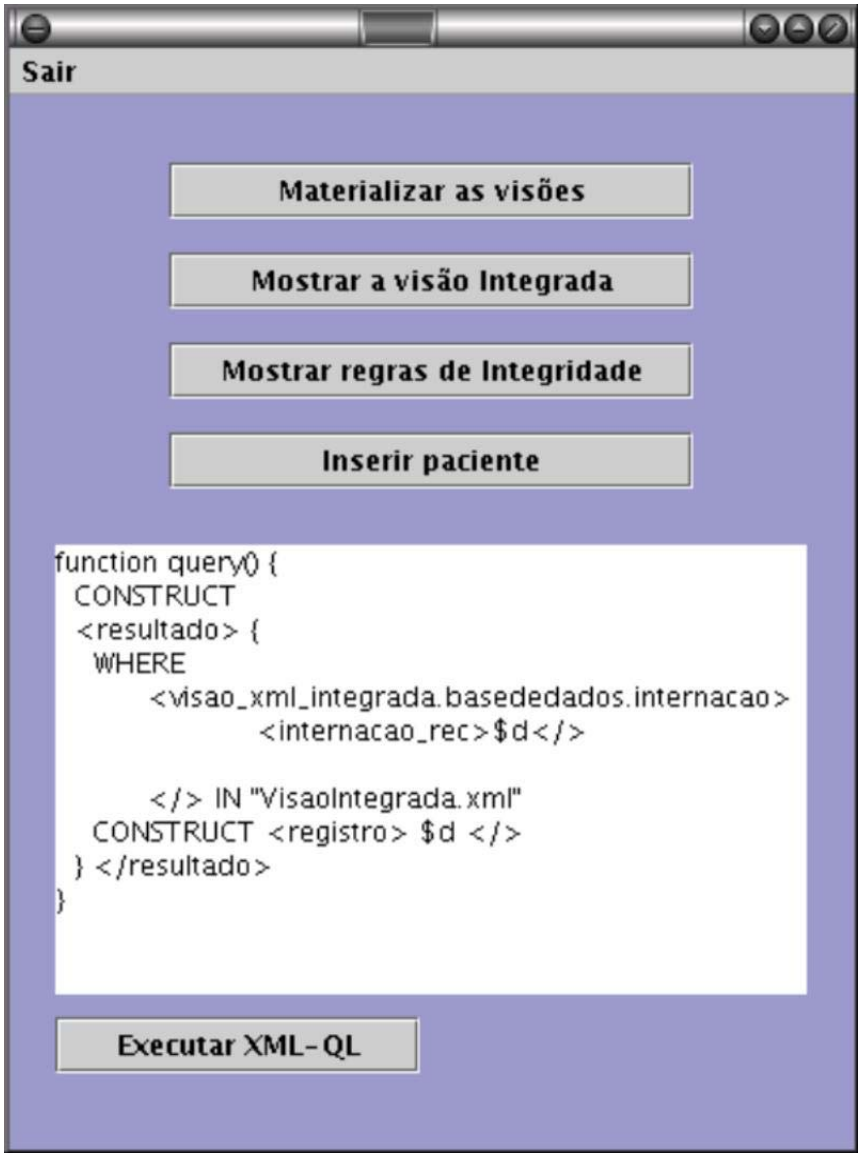

Figura 31 : Tela inicial do protótipo.

- mostrar visão integrada : mostra a visão integrada dos dados através de uma representação gráfica (Figura 32);
- mostrar regras de integridade : mostra as regras de integridade definidas em um arquivo XML previamente por um *expert* humano;

- inserir dados : abre uma interface para inserção de dados na visão XML integrada (Figura 35);
- executar XML-QL : executa a consulta XML-QL definida no campo acima dele localizado sobre a visão XML integrada.

A Figura 32 mostra a visualização dos dados XML armazenados na visão integrada, demonstrados através de uma interface gráfica que mostra a árvore de elementos e a descrição textual dos mesmos.

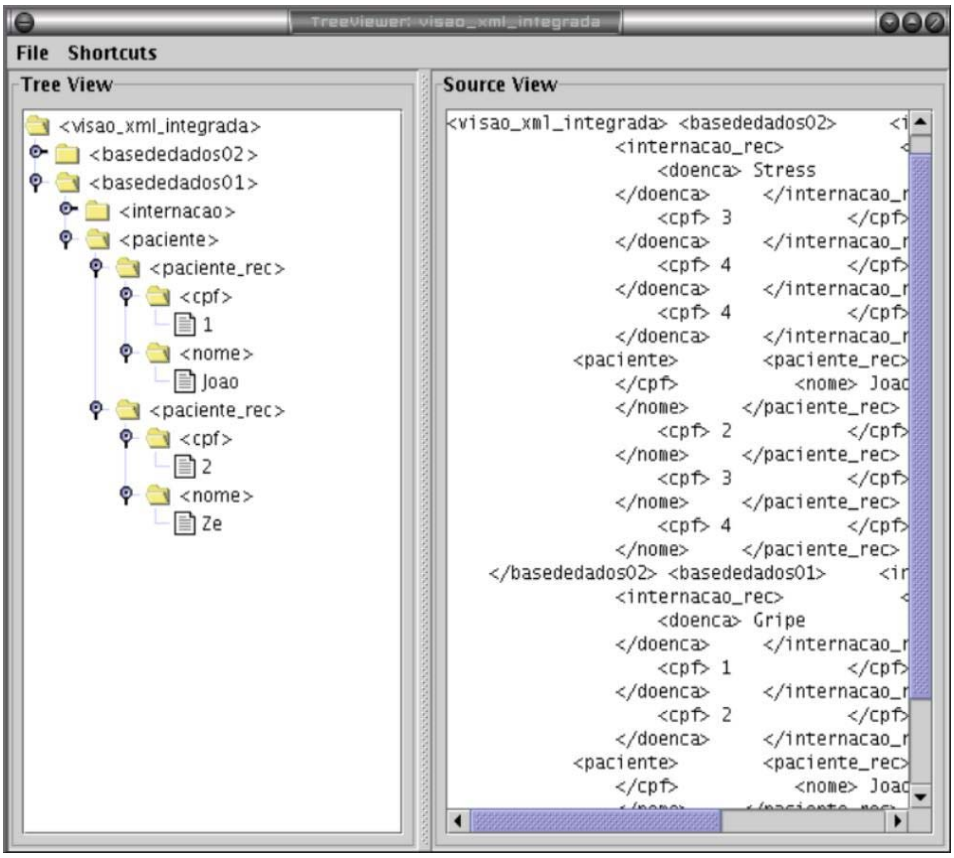

Figura 32 : Visualizando a visão XML integrada dos dados.

A Figura 33 mostra a ocorrência de uma violação de chave primária no sistema, quando o usuário tentou inserir um registro de paciente em uma das bases representadas na visão com cpf = 1 e nome = Marcos, sendo que já existe um registro com cpf = 1 na visão indicando outro paciente.

A Figura 34 ilustra a ocorrência de uma violação de chave estrangeira quando o usuário tenta inserir na visão um registro de internação para o cpf = 33 com a doença =

gripe sendo que não existe no cadastro de pacientes que é chave estrangeira para o cadastro de internações, nenhum registro de paciente com cpf = 33.

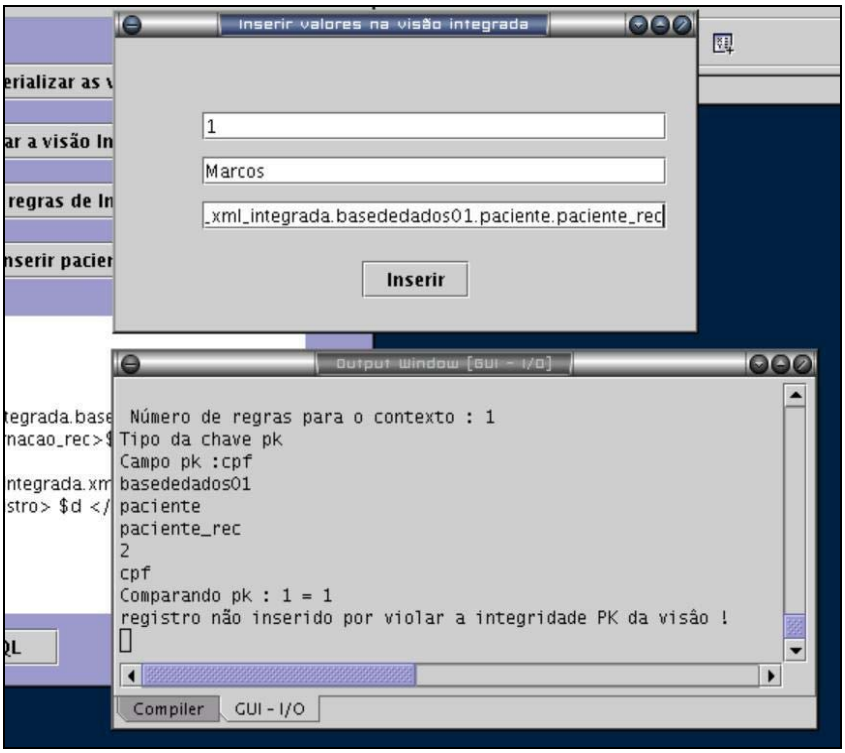

Figura 33 : Exemplo de violação de chave primária.

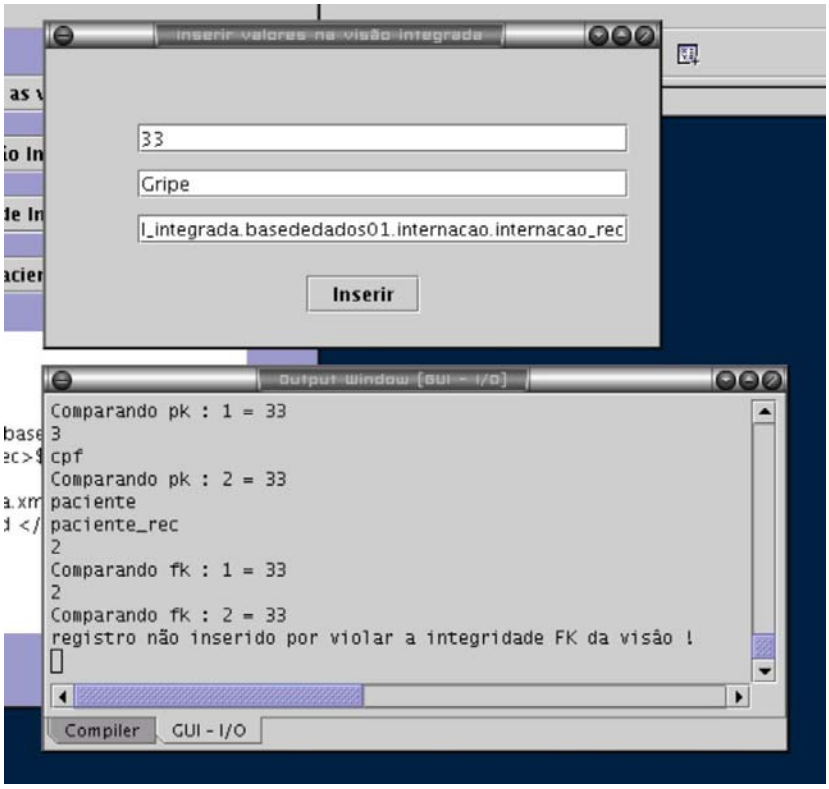

Figura 34 : Exemplo de violação de chave estrangeira.

A Figura 35 ilustra a inserção de um registro de um paciente com cpf = 55 e nome = “Mauri Ferrandin” na visão XML integrada dos dados.

A Figura 36 mostra a execução de uma consulta XML-QL que recupera todos os registros de pacientes de todas as bases que estão integradas na visão XML integrada dos dados.

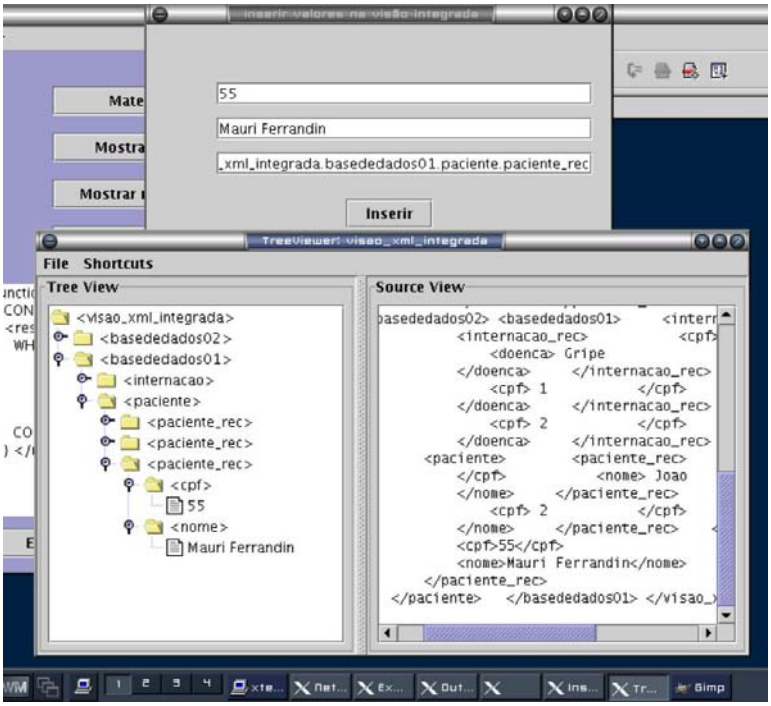

Figura 35 : Exemplo de inserção de um registro na visão XML integrada.

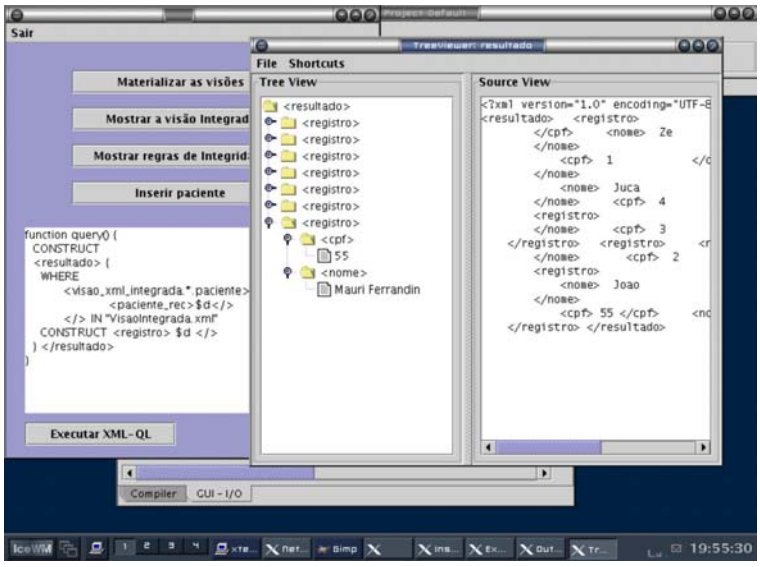

Figura 36 : Consulta recuperando todos os registros de pacientes em todas as bases.

5.3. Comentários finais

Neste capítulo foi documentado a proposta de um sistema para integrar dados de fontes heterogêneas distribuídas. É importante ressaltar que a especificação e implementação de um modelo completo é uma tarefa que demanda muito tempo e trabalho, assim, na implementação do protótipo foi dada uma maior ênfase para as questões chaves em um modelo de implantação, tais como, definição de padrões de formatação dos dados e esquemas para exportação, a disponibilidade de uma meio eficiente de prover o acesso das aplicações aos dados integrados e as questões de integridade em casos de atualização dos dados presentes na visão integrada.

Muitos problemas tiveram que ser contornados, como por exemplo, na proposta inicial do modelo, as atualizações na visão materializada seriam feitas através da própria linguagem de consulta para dados XML, mas com a adoção da XML-QL surgiu o problema da mesma não suportar updates, assim, foi necessário a criação de um mecanismo para alterar a visão XML integrada dos dados utilizando a API DOM, o que limita a performance do protótipo quando a integração envolver grande volume de dados, uma vez que para sua manipulação toda a visão terá que ser carregada para a memória.

6.Análise e Interpretação dos Resultados

Como análise final do sistema proposto, é possível afirmar que o protótipo atendeu aos principais requisitos para os quais ele foi projetado, tais como :

- prover uma maneira eficiente e transparente para os usuários acessarem a dados armazenados em bancos de dados heterogêneos distribuídos;
- especificar um mecanismo de manutenção de restrições de integridade na visão XML integrada dos dados evitando assim a ocorrência de problemas de integridade se os dados nela presentes fossem serializados de volta para as bases.

Ao longo do da fase de desenvolvimento do protótipo foram necessários vários testes com diversas implementações de protótipos de *wrappers*, linguagens para consultas da dados XML, analisadores (parsers) entre outros, e o que se pode perceber é que em muitos casos falta padronização no que diz respeito ao projeto e desenvolvimento de sistemas, bem como, a ausência de um modelo básico mínimo a ser seguido para a criação de um modelo para integração de dados.

Na seqüência serão expostas as principais van*tag*ens e desvan*tag*ens do modelo proposto.

6.1. Van*tag*ens do sistema proposto

O sistema provê uma maneira única de armazenamento e tratamento de informações exportadas das bases relacionais através de XML, assim, todas as informações referentes a dados ou questões de integração poderão ser acessadas de maneira padronizada;

A manutenção das regras de integridade em visões XML exportados de bases relacionais não impõe grandes dificuldade para especificação pelo fato de que os dados armazenados nas bases relacionais já possuem uma estrutura regular com regras de integridade bem definidas;

Para operações que envolvam apenas consultas (sem atualizações na visão XML) a complexidade do sistema se torna muito reduzida.

O protótipo está implementado em linguagem multiplatafoma, o que aumenta a flexibilidade dos sistema não somente no sentido de se utilizar somente padrões independentes de plataforma ou fornecedor para intercâmbio de dados – como é o caso do XML, mas também para criação modelos capazes de rodar nos mais variados ambientes.

6.2. Desvan*tag*ens do sistema proposto

Necessidade da presença de um expert humano para configurar e dar o *startup* inicial no sistema e atualizá-lo cada vez que as fontes de dados forem alteradas.

Complexidade para implementação do sistema levando em conta situações em que ocorram atualizações nos dados da visão XML global tendo assim que propagar estas atualizações para as fontes de dados relacionais.

O sistema tem uma performance muito baixa quando utilizado para integrar bases de dados com grande volume de dados, uma vez que o mesmo utiliza como a API DOM como *core*, assim, a visão integrada precisa ser armazenada por completa na memória para se efetuar qualquer transação, seja ela um simples consulta, ou uma alteração.

Uma grande deficiência dos sistema proposto está em não possibilitar as atualizações na visão XML integrada através de uma linguagem para consulta a dados XML, uma vez que a linguagem escolhida para suportar as consultas das aplicações sobre a visão foi a XML-QL que de acordo com a Tabela 5 : Comparativo entre linguagens de consulta para XML, não suporta atualizações. Assim, toda a vez que for necessário uma aplicação atualizar dados na visão ela terá que fazê-lo através de uma API para manipulação de dados XML tal como o SAX ou o DOM.

7.Conclusões e Perspectivas Futuras

7.1. Resumo

Com o advento da Internet existe uma necessidade crescente de que aplicações façam acesso a bases heterogêneas distribuídas. Essas bases heterogêneas não foram planejadas ou projetadas para suportarem tal integração, assim, diversas barreiras surgem ao se tentar integrá-las, tais como problemas da heterogeneidade semântica e diferenças entre representação de esquemas, sem levar em conta os problemas decorrentes da própria natureza do SGBD, tais como, fabricantes diferentes, arquiteturas e sistemas operacionais, entre outros.

Além de todos estes agravantes, podemos ainda considerar que estes dados integrados serão também atualizados pelas aplicações e teremos então um cenário ainda mais complexo, onde terá que ser levada em conta a propagação destas consultas (atualizações) e também a maneira pela qual serão mantidas as restrições integridade mínimas na camada mediadora para que a propagação de um consulta não venha a causar problemas de integridade nas bases distribuídas.

Esta dissertação discutiu a utilização do padrão XML para integração de fontes de dados heterogêneas mais especificamente bancos de dados relacionais heterogêneos distribuídos, propondo um sistema capaz de integrar dados distribuídos e disponibilizá-los de maneira transparente aos usuários e/ou aplicações.

O escopo deste trabalho envolveu diversas atividades, tais como levantamento bibliográfico sobre o assunto e as diversas linhas de pesquisa correlacionadas, estudo sobre o funcionamento e arquitetura dos sistemas de bancos de dados distribuídos, estudo do padrão XML e seus subpadrões, estudo sobre arquiteturas e técnicas para integração de dados relacionais, estudo da problemática da especificação, manutenção e validação de regras de integridade em dados armazenados através de XML e, por último, proposta de um sistema para integrar os dados de bases distribuídas através de visões, capaz de manter a integridade referencial na visão integrada dos dados em casos de alterações destes (dados) na mesma (visão), de maneira que os dados nela presentes

respeitem as restrições de integridade existentes no banco de dados do qual eles são originários.

Os objetivos propostos com este trabalho foram de maneira geral alcançados, o sistema proposto mostrou-se capaz de integrar fontes heterogêneas de dados através de uma camada de mediação criada entre os dados distribuídos e as aplicações finais, bem como, a proposta de manutenção da integridade referencial dos dados na camada mediadora especificou um esquema que pode ser adotado não somente para questões de integridade na integração de fontes heterogêneas de dados, mas também para definição de regras de integridade em documentos XML.

O padrão XML cai como uma luva para solucionar estes problemas, uma vez que o mesmo é independente de plataforma, é um padrão aberto, e através dele podemos representar qualquer tipo de dado, desde os que se encontram em bases relacionais e/ou objetos, e também dados semi-estruturados.

Muitas ferramentas e APIs para manipulação de dados armazenados através de XML já foram implementadas, facilitando e tornando o padrão XML uma poderosa ferramenta de integração, tais como as APIs SAX e DOM, e as diversas linguagens para consulta da dados XML.

7.2. Comparação com as soluções existentes

Em relação aos outros modelos propostos para integração de fontes heterogêneas de dados, o modelo proposto por esta dissertação incluí no contexto da integração de fontes de dados a problemática de preservação da integridade referencial especificada para os dados em suas bases de origem, problema este não levado em conta na maioria dos sistemas propostos, isto se deve ao fato de que a maioria deles foi criada apenas para integrar fontes de dados XML sem dar um enfoque a integração de dados exportados para XML provenientes de bases relacionais, ou nem cogita a possibilidade de atualização dos dados materializados e conseqüente necessidade de serialização dos mesmos para suas bases de origem.

Outra van*ta*gem do modelo proposto em relação aos modelos estuados é que o mesmo foi implementado utilizando apenas padrões não proprietários e independente de qualquer plataforma de hardware ou software.

7.3. Pontos fracos e pontos fortes

No contexto das pesquisas atuais na área de armazenamento, gerenciamento e integração de dados, é fundamental a proposta de novas soluções capazes de integrar dados independente da sua origem, sejam dados provenientes de um SGBD objeto/relacional, um documento texto, ou até mesmo um *site* na Internet. O padrão XML e os subpadrões a ele correlacionados possuem a flexibilidade necessária para possibilitar o intercâmbio de dados entre sistemas com alto grau de heterogeneidade, e não restam dúvidas de que a combinação das tecnologias XML com ferramentas e APIs de programação modernas tais como Java com capacidade de ser multiplataforma e com os modernos SGBDs hoje existentes irá resultar em um novo modelo de sistema para tratamento, integração, manipulação e intercâmbio de informações.

Como principal deficiência deste modelo, podemos citar a necessidade da presença de um especialista humano para iniciar e solucionar conflitos de integração, bem como definir a integridade a ser mantida nos dados integrados, isto aumenta a possibilidade da ocorrência de erros.

Como ponto forte, temos a utilização do XML para todo o intercâmbio de dados dentro do modelo, exceto é claro na exportação das bases relacionais através dos *wrappers* onde foi necessário utilizar a linguagem de consulta nativa a cada diferente SGBD (SQL na maioria dos casos), toda a integração, definição de restrições, armazenamento de regras de localização estão definidos baseados no padrão XML. Isto torna o modelo mais flexível e possibilita a integração de alguns módulos do mesmo com outros modelos que também sejam baseados no mesmo padrão.

7.4. Perspectivas futuras

Como sugestões para trabalhos futuros, pode-se citar, (i) a especificação de um sistema capaz de automatizar a definição das regras de integridade para a visão integrada dos dados diminuindo a necessidade de intervenção humana; (ii)

implementação de *wrappers* com suporte a SGBDs orientados a objetos; (iii) integração de dados exportados de bases relacionais com documentos ou bases de dados armazenadas através de XML e documentos semi-estruturados; (iv) estudo para verificar a viabilidade da utilização de visões materializadas XML em grande bases de dados; (v) implementação de controle de transações e segurança para o sistema atual; (vi) especificação de um esquema para criação de restrições de integridade em visões de dados XML materializadas implementando outras funcionalidades além da chave primária e chave estrangeira demonstradas neste trabalho.

8.Referências Bibliográficas

[Abiteboul 1997b] ABITEBOUL, S. ***Query*ing semi-structured data.** In: ICDT, 1997. [s.n.], 1997. p.1–18.

[Abiteboul et al., 1997a] ABITEBOUL, S. et al. **Views for semi-structured data.** In: WORKSHOP ON MANAGEMENT OF SEMISTRUCTURED DATA, 1997. **Proceedings...** Tucson, Arizona: [s.n.], 1997.

[Abiteboul et al., 1998] ABITEBOUL, S. et al. **Incremental maintenance for materialized views over semistructured data.** In: PROC. 24TH INT. CONF. VERY LARGE DATA BASES, VLDB, 1998. [s.n.], 1998. p.38–49.

[Abiteboul et al., 2000] ABITEBOUL, S.; BUNEMAN, P.; SUCIU, D. **Data on the *Web* : From relations to semistructured data and XML**. Morgan Kaufmann Publishers, San Francisco, CA, 2000.

[Batini et al., 1986] Batini, C.; Leuzirini M.; Navathe S.B.. **A Compartive Analysis of Methodologies for Database Schema Integration**. ACM Computer Survey, vol. 18, n. 4, 323-364, December 1986.

[Bell e Grimson, 1992] Bell, D.; Grimson, J.. **Distributed Database Systems**. Addison-Wesley, 1992, 2ª ed. Pg. 44-55.

[Bourret, 2001] BOURRET, R. **Xml and databases.** 2001. [s.n.], 2001. p.27.

[Bradley, 1998] BRADLEY, N. **The XML Companion**. Adisson-Wesley, Edinburgh Gate, UK, 1998.

[Bray et al., 1999] BRAY, T.; HOLLANDER, D.; LAYMAN, A. ***Namespaces* in XML.** World Wide *Web* Consortium (W3C), Janeiro, 1999. W3C RecommendationREC-xml-names-19900114.

[Brodie e Stonebraker, 1995] Brodie, M.; Stonebraker, M.; **Migrating Legacy Systems: gateways, interfaces & the incremental approch**. São Francisco, CA: Morgan Kaufmann Publishers, Inc., 1995.

[Buneman et ali., 2001] BUNEMAN, Peter; DAVIDSON, Susan; et al. **Reasoning about Keys for XML (Technical Report).** International Workshop on Database Programming Languages (DBPL), 2001. Disponível em http://db.cis.upenn.edu/DL/absreltr.ps.gz

[Buneman, 1997] BUNEMAN, P. **Semistructured data.** 1997. [s.n.], 1997. p.117–121.

[Buretta, 1997] Buretta, M.. Data replication: tools and techniques for managing distributes information. Wiley Computer Publishing, 1997.

[Buretta, 1997] BURETTA, M.. **Data replication: tools and techniques for managing distributes information.** Wiley Computer Publishing, 1997.

[Castro, 1998] CASTRO, C. E. P. S**.. Integração de Legacy Systems a Sistemas de Bancos de Dados Heterogêneos.** Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Jul. 1998.

[Chen et al., 2002] CHEN, Yi; DAVIDSON, Susan; ZHENG, Yifeng. **Validating Constraints in XML.** Department of Computer and Information Science, University of Pennsylvania, 2002. Disponível em http://db.cis.upenn.edu/DL/validate.ps.

[Date, 1991] DATE, C. J. Introdução a Sistemas de Bancos de Dados. Rio de Janeiro: Campus, 1991.

[Elmasri et al., 1987] Elmasri, R.; Larson, J.; Navathe, S. B.. **Integration Algorithms for Database and Logical Database Design**. Technical Report, Golden Valley, Minn.: Honey-well Corporate Research Center, 1987.

[Fallside, 2000] FALLSIDE, D. C. **XML schema part 0: Primer.** World Wide *Web* Consortium (W3C), Fevereiro, 2000. Working DraftWD-xmlschema-0-20000225.

[Fernandez et al, 1999] FERNANDEZ, M.; TAN, W. C.; SUCIU, D. **SilkRoute: Trading between Relations and XML.** University of Pennsylvania: [s.n.], 1999. Disponível em: < http://db.cis.upenn.edu/RXL/papers/sr.html>. Acesso em: 22 mar 2000.Technical Report.

[Florescu et al., 1998a] FLORESCU, D.; KOSSMANN, D. **A performance evaluation of alternative mapping schemes for storing XML data in a relational database.** 1998. 31 p. p. Relatório técnico.

[Florescu et al., 1998b] FLORESCU, D.; LEVY, A. Y.; MENDELZON, A. O. **Database techniques for the world-wide *Web*: A survey. SIGMOD Record**, [S.l.], v.27, n.3, p.59–74, 1998.

[Furgeri, 2001] FURGERI, S. **Ensino Didático da Linguagem XML**. Érica, 2001.

[Georgakopoulos et al., 1994] GEROGAKOPOULOS, et al. **Using Tickets to Ensure Serializability of Multidatabase Transactions.** IEEE Transactions on knowledge and Data Engineering, vol. 6, n.1, February, 1994.

[Gupta et al., 1995] GUPTA, A.; MUMICK, I. S. **Maintenance of materialized views: Problems, techniques and applications.** IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and DataWarehousing, [S.l.], v.18, n.2, p.3–18, 1995.

[Hull, 1996] HULL, R.; ZHOU, G. **A framework for supporting data integration using the materialized and virtual approaches.** 1996. [s.n.], 1996. p.481–492.

[Hull, 1997] HULL, R. **Managing semantic heterogeneity in databases : A theoretical perspective.** In: ICDT, 1997. [s.n.], 1997.

[Lee e Chu, 2000] Lee, Dongwon; Chu, Wesley W. **Comparative Analysis of Six XML Schema Languages.** University of California, Los Angeles 2000.

[Manica, 2001] Manica, Heloise. **Bancos de dados distribuídos heterogêneos: Arquiteturas, tecnologias e tendências.** Dissertação de mestrado em Ciências da Computação, Departamento de Informática e Estatística – INE, 2001.

[Mello et al., 2000] MELLO, R. S. et al. **Dados semi-estruturados.** In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 2000. [s.n.], 2000. p.39.

[Murphy e Grimson, 1995] MURPHY, J. & GRIMSON, J.. **Multidatabase Interoperability in the Jupiter System.** Information and Software Technology. Vol 37, N. 9, 1995.

[Nestorov et al., 1998] NESTOROV, S.; ABITEBOUL, S.; MOTWANI, R. **Extracting schema from semistructured data.** 1998. [s.n.], 1998. p.295–306.

[Özsu e Valduriez, 1999] Özsu, M. Tamer; Valduriez, Patrick. **Principles of distributed Database Systems**. New Jersey: Prentice Hall, 2ª ed., US, 1999.

[Papakonstantinou et al., 1995] PAPAKONSTANTINOU, Y. et al. **A *query* translation scheme for rapid implementation of *wrappers*.** In: 4TH INTL. CONF. ON DEDUCTIVE AND OBJECT-ORIENTED

[Quan et al., 2001] QUAN, L.; CHEN, L.; RUNDENSTEINER, E. A. **Argos: Efficient refresh in an XQL-based *Web* caching system.** Lecture Notes in Computer Science, [S.l.], v.1997.

[Sheth e Larson, 1990] Sheth, A.P.; Larson, J. A.. **Federated Database Systems for managing Distributed, Heterogeneous and Autonomous Databases.** ACM Computing Surveys, vol. 22, n. 3, Sept, 1990.

[Silva, 1994] Silva, S. D.. **Sistemas de Bancos de Dados Heterogêneos: Modelo de Execução de Gerência de Transações**. Tese de doutorado em informática, Dept. de Informática PUC-Rio. Rio de Janeiro, 1994.

[Silva, 2001] SILVA, A. S. **Materialização de Visões Relacionais para Dados Semi-Estruturados através de Ontologias**. Universidade Federal do Rio Grande do Sul, 2001. Dissertação de Mestrado.

[Widom, 1995] WIDOM, J. **Research problems in data warehousing.** In: 4TH INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 1995. **Proceedings...** Baltimore, Maryland: [s.n.], 1995. p.25–30.

[Wiederhold, 1992] WIEDERHOLD, G. Mediators in the architecture of future information systems. **Computer Magazine of the Computer Group News of the IEEE Computer Group Society**, [S.l.], 1992.

[Wood, 1998] WOOD, L. **Documento object model (dom).** World Wide *Web* Consortium (W3C), Outubro, 1998. W3C working draft.

[Yan et al., 1997] Yan, L. L.. Özsu, M. T.; Liu, L.. **Accessing Heterogeneous Data Through Homogenization and Integration Mediators**. In 2nd Int. Conf. On Cooperative Information Systems (CoopIS97'), 130-139, June 1997.

[Yao et al., 1982] Yao, S. B.. Waddle, V.; Housel, B.. **View Modeling and Integration Using the Functional Data Model**. IEEE Trans. Software Eng., vol 8, n. 6, 544-554, November 1982.

[Zhu, 2000] ZHU, Y. et al. **Materializing *Web* data for OLAP and DSS.** In: *WEB*-AGE INFORMATION MANAGEMENT, 2000. [s.n.], 2000. p.201–214.

9.Apêndice

```
XML-QL ::= (Function | Query) <EOF>
Function ::= 'FUNCTION' <FUN-ID> '(' (<VAR>(':' <DTD>)?)* ')' (':'
<DTD>)?
   Query
'END'
Query ::= Element | Literal |  <VAR> | QueryBlock
Element ::=  StartTag  Query  EndTag
StartTag ::= '<'(<ID>|<VAR>) SkolemID? Attribute* '>'
SkolemID ::= <ID> '(' <VAR> (',' <VAR>)* ')'
Attribute ::= <ID> '='  ('"' <STRING> '"'  |  <VAR> )
EndTag ::= '<' / <ID>? '>'
Literal ::= <STRING>
QueryBlock ::=  Where  Construct ('{' QueryBlock '}')*
Where ::= 'WHERE' Condition (',' Condition)*
Construct ::= OrderedBy? 'CONSTRUCT'  Query
Condition ::= Pattern BindingAs*  'IN' DataSource  |  Predicate
Pattern ::= StartTagPattern Pattern* EndTag
StartTagPattern ::= '<' RegularExpression Attribute* '>'
RegularExpression ::= RegularExpression '*' |
RegularExpression '+' |
RegularExpression '.' RegularExpression |
RegularExpression '|' RegularExpression |
<VAR> |
<ID>
BindingAs ::= 'ELEMENT_AS' <VAR>  | 'CONTENT_AS' <VAR>
Predicate ::= Expression OpRel Expression
Expression ::=  <VAR> | <CONSTANT>
OpRel ::= '<' | '<=' | '>' | '>=' | '=' | '!='
OrderedBy ::= 'ORDERED-BY' <VAR>+
DataSource ::= <VAR> | <URI> | <FUN-ID>(DataSource (',' DataSource)*)
```

Anexo 1 : XML-QL Grammar