SILVIO BORTOLETO

INTEGRAÇÃO DE MULTIDATABASES HETEROGÊNEOS COM APLICAÇÃO DE XML SCHEMAS

CURITIBA

2004

SILVIO BORTOLETO

INTEGRAÇÃO DE MULTIDATABASES HETEROGÊNEOS COM APLICAÇÃO DE XML SCHEMAS.

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática Aplicada.

Área de Concentração: *Metodologias e Técnicas de Computação*

Orientador: Prof.Phd. Edgard Jamhour

CURITIBA

2004

..

TERMO DE APROVAÇÃO

DEDICATÓRIA

Aos meus familiares, em especial à minha esposa Kátia e filha Sophia .pela compreensão da minha ausência durante esta fase.

AGRADECIMENTOS

- Primeiramente a Deus pela minha existência com saúde, proteção e gosto pelo trabalho;
- A toda família pela compreensão da ausência em ocasiões importantes;
- Em especial ao Prof. Phd. Edgard Jamhour, pela orientação e contribuição no desenvolvimento desta;
- Aos colegas de turma, pelo conhecimento adquirido com seus ricos depoimentos e relatos de experiências;
- Aos colegas de trabalho, pelas palavras de incentivo e companheirismo;
- Para todos que, com seu trabalho e senso crítico, contribuíram efetivamente para a confecção deste trabalho.

EPÍGRAFE

“ O REALISTA NÃO MUDA A REALIDADE. O SONHADOR, MAIS CEDO OU MAIS TARDE, ACABA POR FAZER DELA O QUE ELE SONHA”

Autor desconhecido

.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

.

RESUMO

O desenvolvimento de software hoje alcança níveis consideráveis de evolução e portabilidade. Entretanto, ainda são encontradas grandes dificuldades na integração dos bancos de dados, seja por fabricantes diversos ou ainda por que as ferramentas de conectividade não possuam uma flexibilidade para tratar as semânticas necessárias para esta integração. A Integração de Multidatabases heterogêneos tem se tornada peça fundamental no meio governamental e empresarial. A capacidade de executar análises sobre dados posicionados possibilita aos seus usuários obter informações estratégicas. Para integrar diversas fontes de dados heterogêneas, o modelo de dados deve capturar o significado de toda a informação nos sistemas de Banco de Dados componentes. Para que seja concretizada, a integração, um elemento é vital: o metadado, a informação sobre a informação, que define localização, contexto, premissas e restrições de uso dos dados corporativos. Este trabalho propõe a utilização de uma ferramenta que possibilite o tratamento dos metadados das diversas fontes heterogêneas de uma forma dinâmica com a construção de uma API (aplication program interface). Este sistema de integração utiliza um conjunto de correspondências entre os esquemas das fontes de informação, para determinar como as consultas serão respondidas a partir da recuperação dos dados existentes nas fontes locais. Os resultados dessas consultas são traduzidos, filtrados, integrados e apresentados ao usuário, para que ele possa manipular tais informações de maneira que atenda sua necessidade. A representação da integração é feita através da linguagem XML, onde a API ao final do processamento cria um arquivo XML onde os dados estarão contidos e também o seu respectivo XML Schema para a consistência dos dados existentes no arquivo. A linguagem XML permitirá, devida sua flexibilidade, representar fontes com estruturas heterogêneas e também a conversão de qualquer dado para a sua estrutura, assim permitindo a sua manipulação nas mais variadas plataformas existentes.

ABSTRACT

The software development today reaches considerable levels of evolution and portability. However, still great difficulties in the integration of the date bases are found, either diverse manufacturer or still why the connectivity tools do not possess a flexibility to treat semantic the necessary ones for this integration. The Integration of Multi databases heterogeneous if has become basic part in the governmental and enterprise way. The capacity to execute analyses on located data makes possible its users to get strategic information. To integrate several sources of heterogeneous data, the model of data must capture the meaning of all the information in the systems of component Database. So that it is materialize, the integration, an element is vital: the metadata one, the information on the information that defines localization, context, premises and restrictions of use of the corporative date. This work considers the use of a tool that makes possible the treatment of the metadata ones of the several heterogeneous sources of a dynamic form with the construction of a API (application program interface). This system of integration uses a set of correspondences between the projects of the information sources to define as the consultations will be answered from the recovery of the existing date in the local sources. The results of these consultations are translated, filtered, integrated and presented to the user, so that it can manipulate such information thus he takes care of its necessity. The representation of the integration is made through language XML, where the API to the end of the processing creates an archive XML where the data will be contained and also its respective XML Schema for the consistency of the existing data in the archive. Language XML will allow, due its flexibility, to also represent sources with heterogeneous structures and the conversion of any data for its structure, thus allowing its manipulation in the most varied existing platforms.

1 INTRODUÇÃO

Com a evolução e a diminuição do custo dos recursos de informática, tanto em hardware quanto em software, os conceitos da Tecnologia de Informação foram ampliando a abrangência de sua atuação nos negócios das empresas.

As evoluções destes conceitos envolvem diversas tecnologias e produtos de diferentes fabricantes. Um dos maiores problemas de uma organização é que pela complexidade de suas aplicações e a escassez de seus recursos, tanto técnicos quanto humanos, não conseguem manter atualizado todo acervo de aplicações nestas novas tecnologias. Isto trouxe como conseqüência, diferentes aplicações em diferentes estágios de evolução tecnológica. As aplicações que ficam defasadas do estágio atual são denominadas “legadas”. Como o ciclo de evolução da tecnologia é cada vez mais curto, o legado cresce rapidamente nas corporações.

A integração de informações tem sido amplamente abordada pela literatura [SP94]. Assim, vários sistemas têm sido propostos e desenvolvidos para integrar múltiplas fontes. Integrar informações de múltiplas fontes é uma tarefa complexa. Uma das razões para tal complexidade é a heterogeneidade na estrutura das fontes. Uma forma de minimizar a complexidade da integração, adotada pela maioria dos sistemas de integração, é definir um modelo de dados comum para representar a estrutura e conteúdo das fontes.

A Integração de Multidatabases heterogêneos vem se tornando nada peça fundamental no meio governamental e empresarial. A capacidade de executar análises sobre dados posicionados possibilita aos seus usuários obter informações estratégicas que antes só poderiam ser obtidas através de processos manuais cujo tempo de execução era incerto. Mas para que essa visão seja concretizada, um elemento é vital: o metadado, a informação sobre a informação, que define localização, contexto, premissas e restrições de uso dos dados corporativos.

A utilização de uma ferramenta que possibilite o tratamento dos metadados de uma forma dinâmica é a criação de uma API (Aplication Program Interface) que constitui a primeira contribuição deste trabalho. Esta solução envolve diversas áreas de conhecimento na sua concepção, entre elas interoperabilidade e Banco de Dados que pode ser considerado como sendo um dos principais componentes, visto que é

o responsável pelo armazenamento e gerenciamento de todos os esquemas e seus acessos. Pesquisas na área de Bancos de Dados tratam das técnicas de armazenamento e recuperação, interoperabilidade e modelagem. A modelagem dos dados pode ser considerada como uma das mais importantes, pois a qualidade e a funcionalidade de uma aplicação dependem de uma modelagem correta. Hoje existe uma grande quantidade de modelos conceituais propostos, cada qual relativo a um determinado domínio de aplicação, o que não combina com o mundo atual cuja pretensão é a globalização, “padronização” e “interoperabilidade”.

O gerenciamento do metadado está se tornando um ponto de pressão dentro das corporações e conforme suas fronteiras perdem definição em rede. Um movimento que tem reforçado a idéia XML (*Extensible Markup Language*) como meio de integração.

Recentemente, XML (*Extensible Markup Language*) foi proposta pelo W3C como um padrão para representação de informações na Web. Devido à flexibilidade da XML de representar fontes com estruturas heterogêneas e a facilidade de converter qualquer dado em XML. Esta linguagem está sendo proposta também como um padrão para integração de informações.

Desta forma, pode-se fornecer uma representação uniforme e flexível dos dados de fontes distintas. Vários sistemas de integração atuais são baseados e utilizam XML como modelo comum. O sistema de integração utiliza um conjunto de correspondências entre o *esquema global* (esquema canônico) e os esquemas das fontes de informação, chamados *esquemas locais*, para determinar como as consultas serão respondidas a partir de consultas nas fontes locais. Os resultados dessas consultas são traduzidos, filtrados, integrados e apresentados ao usuário, que é a segunda contribuição deste trabalho.

1.1 ORIGEM DO TRABALHO

Atualmente, nas organizações, a informação geralmente está dispersa em diversas fontes de dados, o que dificulta o seu acesso integrado. Na década de 70, muitos sistemas foram desenvolvidos utilizando fontes de dados contidas no *mainframe*. Num segundo momento, conhecido como *dowsizing*, as aplicações

começaram a ser descentralizadas e a utilizar Banco de Dados que processam em máquinas de pequeno porte. Apesar das tentativas de desativar o *mainframe*, na maioria das empresas, isso não foi possível, pois o número de sistemas legados era muito grande. O que se verifica atualmente nas organizações, são ambientes com fontes de dados heterogêneos, onde os dados estão dispersos no *mainframe* e em outras plataformas de servidores de Banco de Dados. Esses ambientes de sistemas heterogêneos de Banco de Dados conduziram as organizações a terem “ilhas de informações”.

Para integrar diversas fontes de dados heterogêneas, o modelo de dados deve capturar o significado de toda a informação nos sistemas de Banco de Dados componentes. Assim, para representar um modelo heterogêneo, o esquema global tem que ter um poder de representação que consiga representar as características específicas dos diversos modelos.

Desde que os sistemas de banco de dados componentes são projetados e implementados independentemente, é normal que existam incompatibilidade entre os dados. O importante é identificar as incompatibilidades para fazer algum determinado tratamento, como nível de abstração (diferentes níveis de detalhes), nome (atributos diferentes com mesmo nome ou atributos iguais com nomes diferentes), tipo, tamanho e outras características que identifiquem o dado.

1.2 MOTIVAÇÃO

O desenvolvimento de software hoje alcança níveis consideráveis de evolução e portabilidade. Entretanto, ainda são encontradas grandes dificuldades na integração dos bancos de dados, seja por fabricantes diversos ou ainda por que as ferramentas de conectividade não possuem uma flexibilidade para tratar as semânticas necessárias para esta integração.

Existem inúmeras empresas enfrentando o problema de como remodelar os seus documentos atuais e integrar seus dados. Para tanto, é necessária uma solução que reduza aos seus componentes atômicos, e que possa combinar e manipular essas peças, separando a apresentação do conteúdo e o conteúdo da estrutura, e, com a mesma importância, deve armazenar esses documentos para

que possam ser apresentados de formas diferentes assim que necessário, ou armazenados em um banco de dados.

A XML está transformando a capacidade da *Web* de um mecanismo de fornecimento de informações interativo para um meio de troca de informações (SIMPSON, 2002; WALSH, 1999). Essa mudança significativa permite que o mundo dos negócios possa ir além da entrega de informações através da *Web*, chegando à condução da missão crítica da interação comercial de toda a empresa, mostrada na figura 1.1.

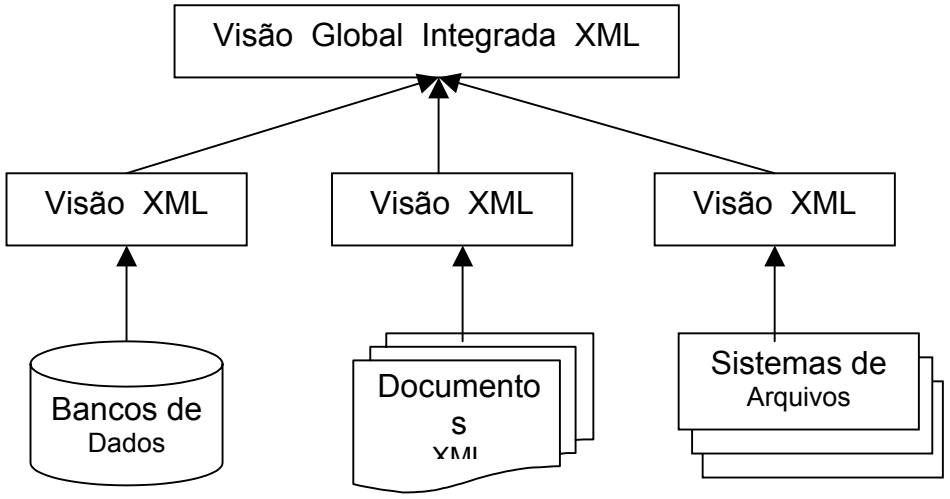


Figura 1.1 Integração de informações

A XML é uma metalinguagem que ajuda na interação dos dados, tornando-os extensíveis, e informando não apenas como exibir os dados, mas sobre os significados deles. A XML oferece um mecanismo independente de plataforma aceito globalmente para gerenciar, armazenar e comunicar informações.

Desta forma, pode-se fornecer uma representação uniforme e flexível dos dados de fontes arbitrárias, como mostrado na figura 1.1. Vários sistemas de integração na *Web* e ambientes corporativos atuais são baseados em mediadores e utilizam XML como modelo comum, os quais são denominados de *XML-Based Mediators* [ABS00, BGL+99,GSN99, OV99, VBO01, VLS0].

Estes sistemas são baseados em uma arquitetura de três níveis de esquema mostrada na Figura 1.1. A visão do sistema de integração utiliza um conjunto de correspondências entre o esquema global os esquemas das fontes de informação, chamados *esquemas locais*, para determinar como as consultas serão respondidas a partir de consultas nas fontes locais. No enfoque materializado, as informações

relevantes são previamente extraídas das fontes de informação, e, posteriormente, traduzidas, filtradas, integradas e armazenadas em um repositório (também chamado de *Data Warehouse*), de maneira que as consultas possam ser avaliadas diretamente neste repositório centralizado, sem a necessidade de acessar as fontes de informação locais.

A heterogeneidade semântica expressa a multiplicidade de possíveis representações de um mesmo conceito do mundo real. Este tipo heterogeneidade pode ocorrer, principalmente, porque projetistas têm diferentes percepções da realidade. Assim, um mesmo conceito do mundo real pode ser representado, por exemplo, por diferentes nomes (heterogeneidade terminológica) e diferentes construtores de modelagem (heterogeneidade estrutural).

A maior parte dos sistemas de integração utilizam os mediadores para selecionar um conjunto de fontes locais que podem ser usadas para responder a uma consulta submetida ao sistema, entretanto falta uma estrutura dinâmica que possa visualizar os metadados e tornar transparente o desenvolvimento de sistemas e aplicações que utilizam multidatabases heterogêneos.

A definição de um dicionário de dados numa API permite especificar de forma precisa as correspondências entre os esquemas locais na integração de sistemas de bancos de dados.

1.3 OBJETIVOS DO TRABALHO

1.3.1 Objetivo Geral

O presente trabalho tem como objetivo geral, propor uma Integração de MultiDatabases heterogêneos com aplicação de *XML Schemas*

1.3.2 Objetivos Específicos

Estudar e apresentar os modelos tradicionais de banco de dados, seu comportamento e aplicações. [OZSU, 2001; ABITEBOUL, 1996; DATE, 2001; SILBERSCHATZ, KORTH, SUDARSHAN, 1999].

Fazer um estudo do desenvolvimento de softwares usando os modelos tradicionais, e como os dados e as informações são tratados. [OZSU, 2001; ABITEBOUL, 1996; DATE, 2001; SILBERSCHATZ, KORTH, SUDARSHAN, 1999].

Apresentar trabalhos desenvolvidos na área de integração de Banco de Dados e comparar com a proposta de um novo modelo de integração para banco de dados heterogêneos [CHANG, 2001; MOULTIS, KIRK, 2000; PFAFFENBERGER, 1999; SILVA, 2001; LIBERTY, KRALEY, 2001; SIMPSON, 2002; WALSH, 1999; REDMOND, 2002].

Apresentar as principais características e aplicabilidade da linguagem XML e a apresentação da semântica para aplicação na integração de banco de dados [CHANG, 2001; MOULTIS, KIRK, 2000; PFAFFENBERGER, 1999; SILVA, 2001; LIBERTY, KRALEY, 2001].

Propor Integração de *MultiDatabases* heterogêneos com aplicação de XML Schemas auxiliado por um dicionário de dados manipulados por uma API.

1.4 ESTRUTURA DO TRABALHO

O primeiro capítulo aborda a origem do trabalho, seus objetivos gerais e específicos, bem como justifica a importância do mesmo.

O segundo capítulo aborda a importância da modelagem de dados e de sistemas descrevendo as principais modelagens tradicionais com as principais características.

No terceiro capítulo, será apresentada um estudo sobre trabalhos e produtos na área de integração de banco de dados comparando com a proposta desta dissertação.

No quarto capítulo, será apresentado o XML como ferramenta a ser utilizada para representação da integração e suas características.

No quinto capítulo, desenvolve-se uma proposta de integração de multidatabases heterogêneos com a criação de um dicionário manipulados por uma API, descrevendo os passos para criação e utilização bem como o tratamento aos dados a serem apresentados como resultado da integração com a utilização do XML.

No sexto capítulo, apresenta-se um estudo de caso da implementação da proposta de integração.

As conclusões e futuros trabalhos são expostas no sétimo capítulo.

Por último, estará listada a bibliografia utilizada para o desenvolvimento da dissertação.

2 MODELAGEM DE DADOS

2.1 TIPOS DE MODELAGEM

2.1.1 Modelo de entidades e relacionamentos

O modelo de entidade e relacionamentos (MER) foi proposto originalmente por PETER CHEN em 1976. Este modelo consagra um método eficiente para representar os dados e ressaltar a diferença entre as estruturas suportadas pelos SGDBs hierárquicos, rede e relacional. Atualmente o MER tem sido utilizado para representar a visão dos dados no projeto de banco de dados. A principal vantagem do MER é a simplicidade. O Modelo de Entidades e Relacionamentos possui apenas três componentes básicos: entidade, atributo e relacionamento (e seus respectivos símbolos para diagramação).

Entende-se que na notação e terminologia, a entidade é a representação genérica de um componente do mundo real, sobre o qual desejamos armazenar informações (atributos). As entidades podem representar coisas tangíveis (pessoal, material, patrimônio,...) ou intangíveis (eventos, conceitos, planos,...). Para notar graficamente uma entidade emprega-se um retângulo identificado por um substantivo (simples ou composto).

Exemplo:

Figura 2.1 – Atributos no MER

Já as regras para atribuição de nomes a entidades, a literatura não consagra um padrão para a atribuição de nomes a entidades, mas, para alcançar um mínimo de padronização, indica-se observar as seguintes regras:

1) Nomes breves e objetivos, grafados em maiúsculas e, que identifiquem facilmente o conteúdo da entidade;

2) No singular, já que a pluralidade decorre, naturalmente, do número de ocorrências (linhas ou tuplas), característica própria de toda entidade.

3) Nomes compostos separados por hífen, eliminando-se o uso de preposições ou outros termos de ligação.

4) Evitar abreviação de nomes. Se necessário, ampliar o tamanho da figura representativa da entidade.

Os atributos são os dados que devemos armazenar a respeito da entidade, para atender às necessidades de informações demandadas pelo usuário. Constituem tudo o que se pode relacionar como próprio (propriedade) da entidade e que, de alguma forma, estejam contidos no escopo do problema em análise. Os atributos qualificam e distinguem as entidades no MER. Em relação ao banco de dados, os atributos representam as colunas, que formam a estrutura de dados das tabelas. As colunas armazenam um valor para cada linha. Esse valor armazenado é designado por valor de atributo. O conjunto de valores de atributos, distintos por um identificador único (chave primária) denomina-se ocorrência. Esse conceito é análogo ao de linha (tupla) em tabela relacional e de registro em arquivo convencional. Pode-se exprimir os atributos no MER, conforme mostrado na figura 2.2 abaixo:

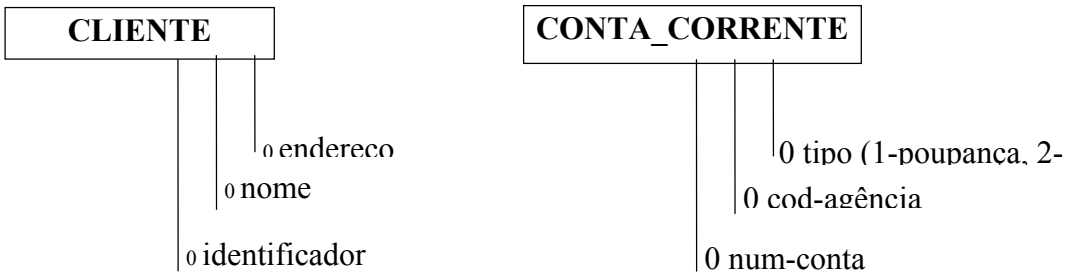


Figura 2.2 – Atributos no MER

Cabe observar, que a representação de atributos no MER pode não deixar claro o gráfico, comprometendo sua objetividade e visão contextual. Esse recurso deve ser reservado para situações especiais, em que você queira destacar um atributo, por considerá-lo elucidativo para o contexto.

Os atributos são usualmente descritos sob a forma de estruturas e elementos no Dicionário de Dados, onde deve constar uma relação de atributos para cada entidade do MER. As notações mais utilizadas para criação de dicionários de dados são as definidas por GANE e YOURDON.

Exemplo da notação de Gane para descrição de estruturas de dados:

CLIENTE
- IDENTIFICADOR
- NOME
- ENDEREÇO

CONTA_CORRENTE
- NUMERO_CONTA
- TIPO-CONTA (1-poupança, 2-c/c)
- AGÊNCIA
  - CÓDIGO_AGÊNCIA
  - ENDEREÇO_AGÊNCIA
- LANÇAMENTOS*
  - NUMERO_LANÇAMENTO
  - DATA
  - TIPO (deb, cre)
  - VALOR

O relacionamento representa a relação existente entre entidades integrantes de um MER. É notado por uma linha ligando as entidades envolvidas e possuem nome e cardinalidade. Veja a figura 2.3 a seguir.

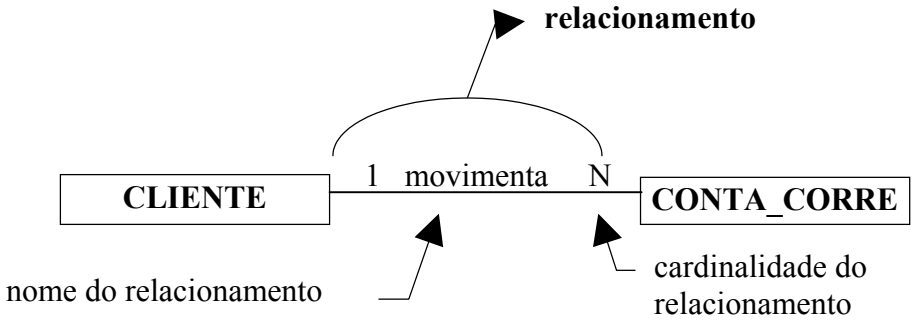


Figura 2.3 – Integrantes de um MER

Peter Chen utiliza um losango para representar o Relacionamento entre entidades, porém, a experiência demonstra que o uso dessa notação contribui para não deixar claro o gráfico e não produz resultado prático, exceto em casos de

relacionamentos que envolvam mais de duas entidades (relacionamento múltiplo - modelo conceitual).

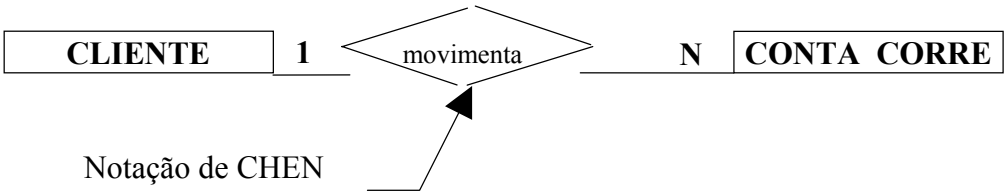


Fig. 2.4 - Relacionamento entre entidades

A cardinalidade constitui um indicativo genérico da quantidade de ocorrências (máxima e mínima) de cada entidade envolvida no relacionamento. É expressa por sinais (flechas, pés-de-galinha, números, letras, e demais), que são grafados sobre a linha do relacionamento, nas duas extremidades do mesmo.

A notação para a cardinalidade é o item que apresenta maior variação entre os autores que escrevem sobre o MER. Neste texto utiliza-se uma barra para notar a cardinalidade “1” e o pé-de-galinha para a cardinalidade “N”. Exemplo:

Considerando a cardinalidade, o relacionamento pode ser de três tipos:

1º) 1:1 - Lê-se UM para UM. Exemplo:

Indica que UMA ocorrência da entidade cliente relaciona-se com UMA ocorrência da entidade conta-corrente e vice-versa.

2º) 1:N - Lê-se UM para MUITOS. Exemplo:

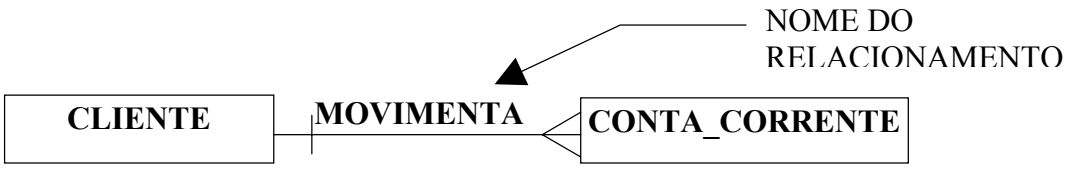
Indica que UMA ocorrência da entidade cliente relaciona-se com muitas ocorrências da entidade conta-corrente e vice-versa.

3º) M:N - Lê-se MUITOS para MUITOS. Exemplo:

Indica que:

- UMA ocorrência da entidade cliente relaciona-se com muitas ocorrências da entidade conta-corrente e;
- UMA ocorrência da entidade conta-corrente relaciona-se com muitas ocorrências da entidade cliente. (cada das contas conjuntas)

O nome do relacionamento é o componente do modelo E-R que identifica o relacionamento, justificando e esclarecendo a importância de sua existência para o contexto estudado. Exemplo:



Nos casos onde o relacionamento é óbvio torna-se dispensável a atribuição de nome ao mesmo. O nome do relacionamento é recomendável nas seguintes situações:

1) Quando existirem diversas possibilidades óbvias de relacionamentos entre o par de entidades, sendo que, deseja-se representar apenas em;

2) Documentação para dar maior clareza ao modelo;

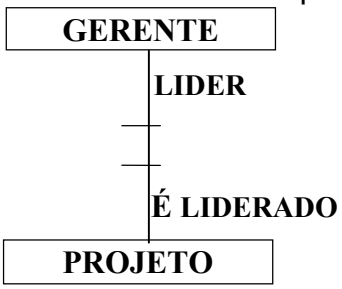
3) Caso ocorra mais do que um relacionamento entre o par de entidades (relacionamento duplo, triplo e superiores);

4) No caso de auto-relacionamentos (entidade relacionando-se com ela mesma-recursividade);

5) Quando da utilização do MER para representar modelos a serem implementados em SGDB rede;

6) Quando da utilização de CASE para desenho do MER (caso a ferramenta obrigue);

O papel das entidades no relacionamento fica implícito no nome e na cardinalidade do mesmo e pode ser inferido a partir desses componentes. Porém, especificar o papel que cada entidade desempenha é uma alternativa, que pode substituir, com maior precisão, a colocação do nome no relacionamento, atribuindo ao MER maior capacidade semântica. Exemplo:



OBS: A maioria das ferramentas CASE não utilizam o PAPEL como alternativa para a construção de MER.

A Integridade Referencial (IR) é notada no MER através de sinalização colocada no relacionamento junto à marca de carnalidade, que indica se o relacionamento é obrigatório ou opcional (total / parcial). Os sinais utilizados para notar a IR variam muito, conforme os autores ou ferramenta CASE adotada e se confundem com a marca de cardinalidade.

No quadro a seguir as notações mais utilizadas para IR:

OPCIONAL	OBRIGATÓRIO	AUTOR
(bolinha aberta)	● (bolinha fechada)	JAMES MARTIN
○ sem marcação	| **(uma barra vertical)**	DIVERSOS
| (uma barra vertical)	|| (duas barras verticais)	DIVERSOS
linha do relacionamento tracejada	linha cheia no relacionamento	BACHMAN / GANE

Tabela 2.1 – Notações mais utilizadas para IR

Exemplo:

São descritas a seguir algumas regras gerais para o modelo:

1) Não podem existir duas entidades iguais no mesmo modelo, ou seja, que representem o mesmo objeto do mundo real e possuam os mesmos atributos; mesmo que elas possuam nomes diferentes.

2) Cada entidade deve possuir pelo menos dois atributos (colunas) e duas ocorrências (linhas).
3) No Modelo Operacional os relacionamentos devem envolver, no máximo, duas entidades.
4) Os relacionamentos Múltiplos do Modelo Conceitual devem ser transformados em entidades em sua passagem para o modelo operacional.
5) Pode existir mais do que um relacionamento entre o mesmo par de entidades (relacionamento duplo, triplo e demais tipos).
6) Para cada relacionamento com cardinalidade “N:N” do Modelo Conceitual, deve ser criada, no Modelo Operacional, uma entidade associativa. Essa entidade será ligada às demais por dois relacionamentos “1:N”, sendo que as cardinalidade “N” de cada relacionamento, serão marcadas ao seu lado.
7) Deve-se avaliar os relacionamentos com cardinalidade “1:1”, verificando se o par de entidades envolvidas pode ser representado por uma entidade única.
8) Cada entidade do MER deve participar de pelo menos um relacionamento. Caso isso não ocorra é provável que a entidade isolada não faça parte do contexto modelado.

São listadas a seguir algumas vantagens do MER:

1) Simplicidade da notação e terminologia
2) Rápida absorção dos conceitos por parte dos técnicos
3) Facilidade de compreensão por parte dos usuários
4) Grande possibilidade de validação do modelo por parte do usuário
5) Capacidade de representar diversos níveis de abstração
6) Compreensão mais objetiva, mais formal e, portanto menos ambígua da do problema.

São listadas a seguir algumas desvantagens do MER:

1) Diversidade da notação e terminologia;
2) Nenhuma ênfase aos processos que manipulam as informações.

Conclui-se que os modelos apresentados tratam domínios e notações específicos e por isto necessitam assistência de pessoas especializadas para ajudar na extração das informações e o reconhecimento correto dos metadados. No caso de diversas fontes de informações a complexidade é exponencial devido à diversidade de notação e nenhuma ênfase aos processos mesmo quando se usam ferramentas automatizadas.

2.1.2 Modelagem de sistemas

Há varias modelagens utilizadas no auxílio ao desenvolvimento de sistemas e que usam técnicas e enfoques diferenciados na sua aplicação. Estas técnicas visam mapear o mundo real através de atividades e fluxos de dados bem como fazer a simulação das mesmas para o refinamento dos modelos de dados e de sistemas.

Na modelagem com fluxo de dados, talvez seja a mais conhecida, pois tem sido utilizada largamente no desenvolvimento da maioria dos sistemas. Os métodos conhecidos como análise estruturada mapeiam o mundo real através de atividades e fluxos de dados. Os problemas com a análise estruturada e, mais recentemente, análise estruturada moderna são relacionados com a dificuldade de conexão do Diagrama de Fluxo de Dados (DFD) com a modelagem de informações e da transição da análise para o projeto. Também, o lado funcional deste enfoque é muito forte e acaba tendo os mesmos problemas da decomposição funcional. A utilização do Diagrama de Entidade e Relacionamento (DER), em conjunto com o DFD e o particionamento do sistema por eventos da Análise Essencial, trouxe alguns resultados positivos, porém ainda continua a dificuldade da conexão prática do DFD e DER e a passagem do DFD, uma representação em rede das atividades e dos depósitos de dados, para o Diagrama Estruturado (DE) com representação hierárquica dos módulos. Ainda, a utilização de notações distintas para funções (DFD) e dados (DER), embora suportada por ferramentas CASE, não é muito bom porque leva a raciocínios separados, distintos.

Outro aspecto a ser mencionado com relação à modelagem com fluxo de dados é o tamanho do dicionário de dados, que tende a ficar excessivo no caso de existirem vários níveis de DFDs, quando centenas de equações de nivelamento de

fluxo de dados podem ser necessárias. Outra consideração é que DFDs não são muito úteis para sistemas ou partes de sistemas que principalmente atualizam ou recuperam dados, ao contrário dos sistemas voltados para o processamento de transações.

Os modelos de dados semânticos (Conceituais), no fim da década de 80, quase todos os SGBD comerciais eram baseados nos modelos hierárquico, relacional, ou de rede. Havia, contudo, várias propostas alternativas de bancos de dados. Uma das primeiras propostas alternativas foi o modelo de dados semântico. A motivação por trás do desenvolvimento dos modelos de dados semânticos (e a maioria dos modelos de dados) é similar à da orientação a objetos: modelar o mundo real tanto quanto possível.

O precursor destes modelos foi o *modelo de entidade-relacionamento* (ELMASRI e NAVATHE, 1984; ELMASRI e NAVATHE, 1986), desenvolvido por Chen na década de 70. Os modelos de dados semânticos, especialmente o modelo ER, são usados primeiramente como ferramentas de projetos de bancos de dados para bancos de dados relacionais ou de rede. Freqüentemente, um esquema (estrutura) de um banco de dados é projetado usando um modelo ER. O esquema semântico é então mapeado em um esquema relacional, usando uma linguagem de banco de dados relacional (ex: SQL ou DML). Isto é, de alguma forma inconveniente e não natural, pois existe uma grande quantidade de mudanças e novas considerações que devem ser levadas em conta neste processo. Muitas mudanças são devidas a características do SGBD que está sendo usado, e outras que afetarão de alguma forma a estrutura do modelo semântico para adaptar-se ao modelo relacional (Normalização, por exemplo, (ELMASRI e NAVATHE, 1984)).

Na decomposição funcional, o domínio do problema deve ser mapeado para uma hierarquia de funções e subfunções. Um exemplo que pode ser citado é o método HIPO (*Hierarchy plus Input-Process-Output*), desenvolvido pela IBM. Baseia-se em dois componentes principais: Representação Hierárquica, que mostra como uma função se subdivide em várias funções, e Representação de Entrada-Processo-Saída, que mostra cada função na hierarquia em termos de suas entradas e saídas. De uma forma geral, na decomposição funcional, o desenvolvedor apresenta como resultado os níveis de sistema, subsistema, função e sub-função.

Os principais problemas encontrados na decomposição funcional são relacionados com a instabilidade da funcionalidade do sistema e com a dificuldade

de se obter alta coesão e baixo acoplamento, na descrição da composição dos componentes do sistema e nas interfaces entre estes componentes. Há dificuldades para identificar funções capazes de suportar os novos requisitos do sistema com mínimas alterações na análise e organização da especificação.

A Análise Orientada a Objetos (AOO) usa a decomposição funcional, porém num contexto muito específico, quando se deseja dividir um grande serviço (comportamento que um objeto deve apresentar) em partes menores para torná-lo mais claro e diminuir a complexidade. Nesta definição é bom ressaltar, que é feita dentro de um contexto muito limitado, não sendo usada como estrutura organizacional principal da análise.

Na modelagem para simulação discreta, qualquer componente no sistema que requeira uma representação explícita em um modelo é chamado de entidade (SEVERANCE, ENBODY, PURDY, 1994). Existem vários tipos de entidades, cada um tendo várias características ou atributos. Embora possam estar envolvidas em diferentes tipos de atividades, pode ser conveniente agrupar-se as entidades baseando-se em um atributo comum. Grupos de entidades são chamados arquivos ou conjuntos.

O objetivo de um modelo para simulação discreta é reproduzir as atividades das entidades engajadas e, a partir daí, conhecer sobre o comportamento e desempenho do sistema. Isto é conseguido quando definimos os estados do sistema e construímos as atividades que alterem esses estados. O estado de um sistema é definido em termos de valores numéricos dados aos atributos das entidades. Um sistema está em determinado estado quando todas as suas entidades estão em estados de sintonia com o domínio dos valores dos atributos que definem aquele estado. A maioria dos sistemas possui um grande número de possíveis estados e a simulação descreve as mudanças entre os estados do sistema. Se o sistema alcança um estado, e a partir deste estado nenhuma mudança é possível, o estado é chamado de *estado fixo*. Estados que não são fixos são conhecidos como *estados transientes*.

Em simulação discreta, o estado do sistema só pode mudar nos tempos de eventos. Uma vez que o estado do sistema permanece constante entre tempos de

eventos, uma descrição completa do estado do sistema pode ser obtida avançando o tempo simulado de um evento ao outro. Este mecanismo é usado na maioria das linguagens para simulação discreta.

A formulação de um modelo para simulação discreta pode ser realizada de três formas:

a) pela definição das mudanças nos estados que podem ocorrer em cada tempo de evento;
b) pela descrição das atividades nas quais as entidades do sistema se envolvem;
c) pela descrição dos processos através dos quais as entidades do sistema fluem.

Pode-se dizer que um evento acontece em um ponto isolado do tempo, no qual decisões devem ser tomadas de forma a iniciar ou terminar uma atividade. Um processo é uma seqüência ordenada de eventos e pode englobar várias atividades, que, por sua vez, correspondem a operações que causam mudanças no estado do sistema [SEVERANCE, ENBODY, PURDY, 1994].

Estes conceitos levam a três alternativas ou enfoques de modelagem para simulação discreta:

a) Modelagem orientada a evento.
b) Modelagem orientada ao exame da atividade.
c) Modelagem orientada a processo.

Para certos tipos de problemas, o uso desta abordagem provê um modelo bem conciso. O modelo é particularmente adequado para situações onde a duração da atividade é indefinida e determinada pelo estado do sistema, satisfazendo uma condição preestabelecida. Entretanto, devido à necessidade de se escalonar todas as atividades a cada avanço do tempo, o método é ineficiente quando comparado com outras abordagens [SOARES, 1992; MAC, 1975].

Já na simulação orientada a processo, muitas estruturas de modelos para simulação incluem seqüências de eventos as quais ocorrem em padrões definidos (por exemplo, uma fila de entidades esperando por um servidor). A lógica associada com tal seqüência de eventos pode ser generalizada e definida por uma única

afirmação. Uma simulação orientada a processo emprega tais afirmações para modelar o fluxo das entidades no sistema. Estas afirmações definem uma seqüência de eventos que é automaticamente executada pela simulação. A simplicidade da simulação orientada por processo reside no fato da lógica dos eventos associada às afirmações estar contida na linguagem de simulação. Entretanto, como estamos restritos a um conjunto de afirmações padrão fornecido pela linguagem, a flexibilidade não é tão grande quando comparada com a modelagem orientada a evento [SOARES, 1992]. Basicamente, uma simulação orientada a processo segue os seguintes passos:

a) definir as entidades do sistema;
b) criar um processo para cada entidade descrevendo suas etapas;
c) executar concorrentemente os processos.

O uso generalizado de simulação como uma ferramenta de análise de sistemas deu origem a uma série de linguagens especificamente projetadas para este fim. Estas diversas linguagens e pacotes de modelagem impõem uma certa estruturação nos modelos de dados e simplificam suas soluções.

Pode-se concluir que as várias modelagens utilizadas no desenvolvimento de sistemas estão focadas na funcionalidade, semântica, fluxo de dados, simulações dos eventos, atividades e processos cuja aplicação depende da necessidade específica dos sistemas por isso identificamos a necessidade de uma integração conceitual para tornar o modelo de integrado inteligível.

3 INTEGRAÇÃO DE BANCO DE DADOS

3.1 XML INTEGRATOR

Esta ferramenta tem como propósito fornecer uma solução para acesso integrado a base de dados heterogêneos baseado nos mapeamentos dos *schemas* dos bancos de dados e em documentos XML.

O XML Integrator é “utilizado na extração e mapeamento de esquemas conceituais locais, tanto de bancos de dados como de documentos XML”. Essa solução, através de um metamodelo XML, une dados semi-estruturados em um acesso integrado a bases heterogêneas.

Os dados a serem usados pela ferramenta são primeiramente persistidos em uma base de conhecimento composta por um banco de dados relacional. Nesse banco há todas as informações internas necessárias à ferramenta. Ele possui também um conjunto de classes de documentos XML armazenando informações dos esquemas de exportação, equivalências, conflitos e mapeamentos entre as fontes de dados. Um wrapper (empacotador) específico para o tipo da fonte de dados (estruturado ou semi-estruturado) realiza a conversão e disponibilidade para acesso local das informações.

Para a extração dos esquemas conceituais de documentos semi-estruturados, deve-se primeiro haver uma instância do documento a ser analisado. O usuário, então, será capaz de modificar ou incluir novas informações que possam não ter sido obtidas durante o processo de extração. O próximo passo consiste de submeter todos os esquemas conceituais ao otimizador de esquemas que realizará o processo de combinação, excluindo as entidades e atributos com o mesmo nome. Assim um esquema conceitual único é obtido, representando uma classe de documentos XML.

As seguintes regras são aplicadas durante o processo de extração do esquema conceitual [ABITEBOUL, CLUET, MILO, GAROFALAKIS, GIONIS, RASTOGI, SESHADRI, NESTOROV, PSAILA]:

Ação	Descrição
Criação do elemento raiz do EE	O elemento raiz do metamodelo é criado a partir do elemento raiz do documento XML. Informações adicionais são fornecidas pelo usuário.
Criação da primeira entidade	Quando o elemento raiz possui componentes de conteúdo “textOnly”, “mixed” ou atributos, então a primeira entidade criada terá o nome do elemento raiz.
Análise de componentes “elementOnly”	Para todo componente do tipo “elementOnly” é criada uma entidade correspondente.
Análise de componentes do tipo “empty”	Os componentes do tipo “empty” são desconsiderados, podendo receber tratamento de objetos externos em futuras.
Análise de componentes do tipo “textOnly”	Para todo componente do tipo “textOnly” é criado um atributo para o componente de nível superior.
Análise de componentes do tipo “mixed”	Para todo componente do tipo “mixed” é criada uma entidade correspondente e o conteúdo texto do componente é incluído como informação adicional à entidade.
Análise dos atributos dos componentes	Para todo atributo de um elemento é criado um atributo para a entidade correspondente.
Identificação de domínios e outras informações	Informações complementares para entidades e atributos e restrições de domínio como intervalos válidos, precisão numérica, unidades, comentários e estrutura de dados, podem ser adicionadas através de informações fornecidas pelos usuários. Os tipos de dados dos atributos são identificados através da utilização de funções de reconhecimento de domínio como “IsNumeric”, “IsText”, “IsBoolean”, etc.

Tabela - 3.1: regras para extração do esquema conceitual

Para mais informações sobre os componentes deve-se consultar [DAR03, RIB03].

Esta é uma ferramenta gráfica bem indutiva que pode ser bem aplicada a bases de razoável complexidade. Propõe a idéia de se utilizar um único metamodelo XML para representar as bases de dados. Além disso suporta a integração física das classes de documentos XML além da lógica, o que pode ser interessante a processos de data mining. Suporta também meios para enriquecimento semântico, o que é carente em algumas demais ferramentas com o mesmo objetivo.

O XMLIntegrator é mais frequentemente utilizado em ambientes onde são encontradas limitações de expressividade. Necessita que todos os processos sejam implementados por interfaces as quais direcionam o usuário, seguindo todos os passos. Depois de a conexão ser estabelecida a extração do esquema conceitual local é realizada e é apresentada ao usuário através de um módulo de edição e geração de esquemas de exportação.

Por meio da análise do dicionário de dados e importações ocorre a extração de esquemas de bancos de dados. Desta forma, as tabelas serão representadas como entidades e as colunas como atributos.

No caso de documentos semi-estruturados, o início da extração é composto por disponibilizar uma instância da documentação. Caso a instância tenha uma associação com DTD, será possível que se realizem modificações ou inclusões de informações previamente não capturadas.

Este modelo sugere o uso de modelo de dados canônico, usado para representar dados estruturados e não estruturados.

As principais vantagens desta abordagem são [DAR03, RIB03]:

- a representação proposta é de fácil entendimento, pois utiliza uma estrutura hierárquica, com expressividade equivalente ao modelo de dados relacional, incluindo ainda aspectos de orientação a objetos adotados pelos SGBDOO;
- fornece uma visão clara da estrutura do banco de dados ou documento, independentemente do modelo de dados adotado localmente;
- a representação é armazenada em formato texto, permitindo assim a portabilidade e interoperabilidade entre os sistemas componentes;
- fornece uma representação flexível, permitindo a inclusão de novas extensões, informações e restrições de dados;
- a representação conceitual de documentos XML simplifica a inclusão de fontes de dados semi-estruturadas em metodologias de acesso integrado a bancos de dados heterogêneos;
- permite uma representação conceitual única para uma classe de documentos semi-estruturados conceitualmente equivalentes, mas estruturalmente diferentes.

3.1.1 Managing the Evolution of XML-based Mediation Queries

Este trabalho é mais focado na manutenção da integração realizada nas várias bases, ou seja, em como manter o esquema integrado. Foram sugeridas algumas etapas para esse sistema [AMO03]: recuperação do esquema atual da

base, atualização do esquema integrado, atualização das consultas disponibilizadas para os usuários.

Aborda como tratar *queries* que dependam de esquemas em XML no caso destes sofrerem alterações, como em sistemas mais dinâmicos.

O seu escopo é mais voltado, então, em como gerenciar o crescimento e mudanças nas estruturas de bases de dados integradas.

3.1.2 Ferramenta para extração de esquemas de bases de dados relacionais

Uma ferramenta com o intuito de fornecer uma única visão das várias fontes dos dados (remotas ou locais), cujo escopo é a implementação da recuperação do esquema atual da base (*lookup)*.

O principal aspecto do *lookup* é a obtenção de um documento XML *Schema* capaz de representar o esquema das diversas fontes de dados. Segundo [LÓS03], o *lookup* deve ser apto a obter esquemas de qualquer tipo de base de dados estruturados ou semi-estruturados. A ferramenta em questão escolheu como Sistemas Gerenciadores de Banco de Dados (SGBDs) o Oracle e o SQLServer.

Características que não sejam desejadas podem ser retiradas do XML *Schema* através do pré-processamento, numa etapa que ocorre no *Conceptual Schema Manager [*LÓS03*].*

A ferramenta foi desenvolvida em Java, e na utilização de APIs padrões na linguagem é possível extrair informações específicas sobre as bases de dados, os metadados [DAT04]. Para os metadados pode-se usar diferentes coleções e técnicas para combinar estruturas, restrições e operações usadas para descrever dados [FAN91].

Uma vez que os metadados já tenham sido obtidos, o próximo passo é obter os dados desejados e então gerar-se o XML *Schema*.

3.1.3 Mapping XML and Relational Schemas with CLIO

CLIO é uma ferramenta para mapeamento semi-automático de esquemas de bases de dados. CLIO possui um mecanismo de mapeamento que permite mapeamento de e para esquemas relacionais e XML. Para que se mantenham as associações dos dados utiliza-se das restrições (constraints) dos dados.

A criação e interpretação dos mapeamentos não consistem de um processo trivial e muitas ferramentas não conseguem implementar esta funcionalidade [DAT04]. Desta forma, o usuário é que deve ser o responsável por identificar manualmente e especificar os detalhes dos mapeamentos, como, por exemplo, geração de *keys,* referências, condições de *joins,* etc.

O propósito do CLIO é que o usuário não precise escrever *queries* complexas ou programas para tradução, mas sim que utilize uma ferramenta de mapeamento de esquemas de alto-nível. O mapeamento é então realizado usando-se de correspondências de valores. Basicamente, essas correspondências especificam como valores para um atributo-alvo são gerados por um ou mais atributos-fonte. Dado esse mapeamento de alto-nível, o mecanismo do CLIO é capaz de “descobrir” uma implementação daquele mapeamento como uma *query* (por ex., SQL). Ou seja, esta ferramenta compila o mapeamento definido em alto-nível (a correspondência de valores) em uma representação de baixo-nível (uma *query).*

Essa compilação ocorre em dois passos: na tradução semântica, as correspondências dos valores são inferidas, ou seja, um mapeamento lógico e preciso deve ser criado. No passo da tradução de dados, esse mapeamento lógico é convertido finalmente em uma *query* capaz de capturar os dados. Mais detalhes sobre o algoritmo desse mecanismo de mapeamento pode ser encontrado em [FAN91].

Na figura 3.1[POP03] pode-se observar graficamente uma representação de como CLIO funciona num mapeamento relacional para XML.

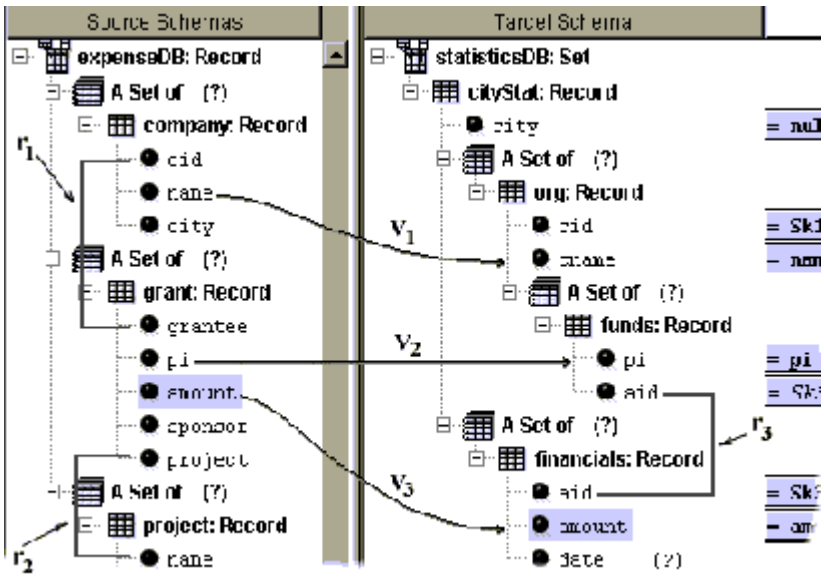


Fig. 3.1: mapeamento de relacional para XML

3.1.4 Common Object Request Broker Architecture (CORBA

Common Object Request Broker Architecture (CORBA) especificado pela OMG (Object Management Group) fornece uma arquitetura independente de plataforma e de linguagem para o desenvolvimento de aplicações distribuídas. Os objetos CORBA podem residir no mesmo processo, na mesma máquina ou em qualquer lugar do mundo.

O paradigma básico do CORBA é a requisição de serviços de um objeto distribuído. Os serviços que um objeto provê são dados pela sua interface que é definida na IDL (*Interface Definition Language*) da OMG. Os objetos distribuídos são identificados por referências aos objetos, que por sua vez são definidos pelas interfaces IDL.

Um dos objetivos da especificação CORBA é que as implementações dos clientes e dos objetos são portáveis. Essa especificação define uma API para clientes dos objetos distribuídos assim como uma API para a implementação de um objeto CORBA. Assim, o código escrito para um produto CORBA de uma determinada organização pode, com facilidade, ser reescrito para funcionar com um produto de outra organização. Mas a realidade no mercado atualmente é que os clientes CORBA são portáveis mas as implementações dos objetos precisam de algum re-trabalho.

O modelo CORBA é baseado em objetos e permite que estes sejam ativados remotamente através de um elemento chamado ORB (Object Request Broker). Este elemento fica situado entre o objeto e o sistema operacional, no qual são adicionadas novas funcionalidades que permitam a comunicação na rede.

A versão 2.0 do CORBA focou na interoperabilidade. Para tanto, essa nova versão definiu um protocolo de Rede chamado *IIOP (*Internet Inter-ORB Protocol) [COM03]. Esse protocolo permite que clientes utilizando um produto CORBA de qualquer vendedor possa se comunicar com objetos usando CORBA de qualquer outro. IIOP trabalha através da Internet, ou mais especificamente, através de qualquer comunicação TCP/IP.

O padrão CORBA define um conjunto de serviços distribuídos para suportar a integração e interoperabilidade de objetos distribuídos. Os serviços são definidos como objetos padrões CORBA com suas interfaces IDL devidamente. Há vários serviços [COF03] CORBA e na tabela 3.2 na página seguinte pode-se visualizá-los com uma breve descrição.

Serviço	Descrição
Object life cycle	Define como objetos CORBA são criados, removidos, movidos e copiados
Naming	Define como objetos CORBA podem ter nomes simbólicos amigáveis
Events	Separa a comunicação entre objetos distribuídos
Relationships	Fornece relacionamentos entre objetos CORBA
Externalization	Coordena a transformação de objetos CORBA para e de mídias externas
Transactions	Coordena acesso atômico a objetos CORBA.
Concurrency Control	Fornece serviço de lock para objetos CORBA para assegurar acesso serializável.
Property	Suporta a associação de pares nome-valor com objetos CORBA.
Trader	Procura de objetos CORBA baseado em propriedades descrevendo o serviço oferecido pelo objeto.
Query	Fornece queries em objetos

Tabela 3.2 - Serviços Corba

Na tabela 3.2 pode-se observar basicamente a comunicação no modelo CORBA. Aprofundando-se um pouco mais na IDL abordada anteriormente, vale ressaltar que a IDL é uma Linguagem de Definição de Interfaces independente de linguagem. Cada linguagem que suporta CORBA tem seu próprio mapeamento IDL. No exemplo apresentado, mostra-se uma implementação da IDL em Java com o intuito de ilustrar como funcionaria na prática [GRE98].

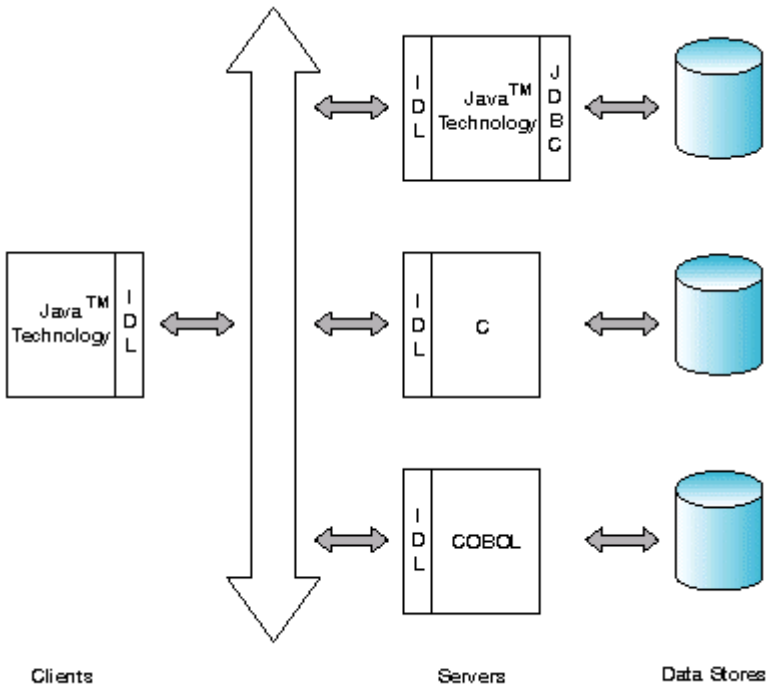


Figura 3.2: visualização da comunicação CORBA (IDL)

O CORBA é uma tecnologia integradora, capaz de se comunicar com diferentes bases de dados através de diferentes tecnologias, plataformas e linguagens. Portanto é uma forma de se obter a integração entre as bases de dados, mas somente no que tange à comunicação. Seria necessário ainda ter a definição dos esquemas das bases de dados e escrever código para o acesso, recuperação e atualização das bases.

3.2 DCOM E COM

Distributed Component Object Model (DCOM) é um protocolo que permite que componentes de softwares se comuniquem diretamente através da Rede. Pode ser usada em vários tipos de transporte, incluindo protocolos da internet como HTTP. Foi baseado na especificação DCE-RPC da *Open Software Foundation*.

O DCOM extende a COM [MIC03] para permitir a comunicação entre objetos em diferentes máquinas. Desta forma a aplicação pode ser distribuída em diferentes localizações. DCOM controla os detalhes de baixo nível dos protocolos de redes e o desenvolvedor pode focar mais no negócio em si. A COM define como os componentes e seus clientes interagem e essa interação é de tal forma que o cliente e o componente podem se conectar sem a necessidade de qualquer sistema intermediário. O cliente chama métodos no componente sem qualquer *overhead* (repetição, como de cabeçalhos por exemplo). Além disso é independente de linguagem.

Basicamente a principal diferença entre COM e DCOM é que a segunda é capaz de trabalhar de modo distribuído. Os desenvolvedores COM podem facilmente aplicar seus conhecimentos a aplicações distribuídas baseadas em DCOM.

Tanto CORBA como DCOM fornecem transparência para localização de objetos e sua implementação, sendo necessário então uma definição de interfaces entre componentes. Portanto os objetivos a serem alcançados com CORBA podem

também ser alcançados utilizando-se DCOM. O DCOM pode ser útil para a integração de sistemas heterogêneos, mas da mesma forma que CORBA, como citado anteriormente, não existe um prévio mapeamento de dados nem de esquemas das bases. Portanto, essas tecnologias servem de apoio para comunicação na integração.

3.2.1 Translating Web Data

Em [POP02] é proposto um *framework* para mapeamento entre qualquer combinação de XML e esquemas relacionais. Um mapeamento em alto-nível definido pelo usuário é traduzido para *queries* que transformam os dados-fonte na representação-alvo. O algoritmo de tradução foi implementado utilizando o CLIO á abordado nesta seção.

O propósito era de criar uma ferramenta capaz de gerar de forma rápida e correta traduções entre dados oriundos da Web. A ferramenta também deveria ser capaz de capturar as semânticas embutidas em estruturas aninhadas.

Entre algumas das características interessantes está a implementação de novos algoritmos para mapeamento entre esquemas e tradução de dados entre esquemas aninhados com regras referenciais aninhadas. As traduções são realizadas baseando-se nas semânticas dos esquemas. O framework garante que a instância-alvo criada satisfaz a estrutura e as regras do alvo, mesmo quando essa instância precisa conter dados que não são oriundos da fonte.

Os mapeamentos lógicos produzidos são dependências do tipo “fonte-alvo” relacionadas a uma query da fonte (como os dados da fonte serão obtidos juntos) e uma query para o alvo/destino (como os dados serão re-estruturados no destino). Essas dependências podem ser usadas num cenário de integração de dados onde o destino é virtual e as queries são respondidas usando a instância da fonte [LE02].

Este framework mostra-se bastante flexível e genérico, uma vez que consegue efetuar a tradução de dados complexos e de forma consistente, como foi descrito anteriormente. Pode ser bastante útil para integrar dados heterogêneos das bases de dados e da web (semi-estruturados). Informações detalhadas podem ser encontradas em [POP02].

3.2.2 Projeto TOX

O *Toronto XML Server* é um projeto cujo objetivo é criar um repositório para dados e metadados XML, com a capacidade de trabalhar com documentos XML reais e virtuais. Os documentos reais são armazenados como arquivos ou mapeados em bases de dados relacionais ou de objetos (de acordo com a estrutura) [TOR03]. Os documentos virtuais podem ser documentos remotos ou *queries.*

Existe um catálogo do sistema com metadados para os documentos, especialmente o esquema, usado no processamento da *query* e otimização. As *queries* podem abranger tanto o catálogo como os documentos e múltiplas linguagens de *query* são suportadas [TOR03].

Entre algumas características mais interessantes do TOX encontram-se a seleção automática de coleções apropriadas para armazenar um documento de acordo com os DTDs, os quais podem ser associados às coleções. Outra característica é a de armazenar documentos como arquivos-texto ou mapeá-los em tabelas relacionais num SGBD, no caso o DB2. Além disso há uma estrutura de dados especialmente definida para acesso rápido. Mesmo com todos esses mecanismos de armazenamento há uma interface comum para navegação.

3.2.3 Projeto Ozone

É um sistema gerenciador de banco de dados orientado a objetos completamente desenvolvido em Java. Inclui implementação da DOM, o que permite

que se persista dados em XML. Pode-se usar qualquer ferramenta XML para obter e acessar esses dados.

O objetivo do Ozone é servir como um SGBD e houve uma preocupação em realizar a persistência inclusive em XML. Desta forma pode ser uma alternativa na integração de dados, uma vez que, sendo possível armazenar as informações em XML pode auxiliar o processo de integração. Esse auxílio ocorre no sentido de que pode não ser necessário fazer o mapeamento relacional, o que é uma tarefa custosa muitas vezes.

3.2.4 Using Correspondence assertions for specifying the semantics of XML-based mediators

Mediadores fornecem uma visão integrada em múltiplas fontes de informação e permitem que queries sejam executadas nessa visão integrada. Em [VLS01] foi proposto o uso de *correspondence assertions* (afirmações de correspondências) para definir a relação entre o esquema do mediador e o esquema das bases de dados.

As *correspondence assertions* então seriam usadas para definir as semânticas dos mediadores em XML. Definindo-se os mediadores assim em alto-nível tem-se a vantagem da indepêndencia de linguagem e melhor entendimento das semânticas associadas ao mediador.

Em sistemas de ingração de dados baseados em XML, uma *mediated view* (visão mediada, ou seja, uma visão dos dados usando um mediador) é definida numa linguagem declarativa especificadamente designada para XML. A visão mediada é baseada em queries, restrições e regras que descrevem como as queries no esquema mediado serão mapeados em queries executadas nas fontes de dados.

Para a geração das visões mediadas pode-se utilizar as *correspondence assertions* e, conforme apresentado em [VID01, LOS01], a visão mediada seria capaz de implementar corretamente a especificação do mediador e tratar problemas

de semânticas heterogênicas. Esse problema é relativo a uma visão ser representada de maneira diferente nas fontes de dados [HUL97].

As semânticas do mediador podem também ser usadas para automatizar outros aspectos da integração dos dados, como na manutenção da visão. Em determinados ambientes mais dinâmicos como a Web, as fontes dos dados podem mudar não somente os dados mas também suas capacidades [VID01, LOS01]. Desta forma, quando um esquema local muda, a definição da visão precisa ser atualizada para refletir as alterações. As *correspondence assertions* do mediador podem ser usadas para re-definir uma visão de acordo com as mudanças nos esquemas locais.

3.2.5 A abordagem POESIA para a Integração de Dados e Serviços na Web Semântica

A Web Semântica [SEM03] consiste em informações que possam ser mais eficientemente re-usadas e transferidas com base no uso de metadados e processamento semântico.

POESIA é uma proposta para a integração de dados com o intuito de contribuir no avanço das aplicações científicas no contexto da Web Semântica.

Entre as principais características pode-se citar: organização de diversas coleções de serviços, seleção de dados e serviços de acordo com seus escopos de utilização, checagem de consistência semântica e estrutural da composição de serviços na web. Essas funcionalidades são implementadas através do uso de ontologias de domínio com múltiplas dimensões.

3.2.6 Discovering View Expressions from a Multi-Source Information System

Existem sistemas de informações compostos por fontes de dados independentes e diversas queries nessas fontes. A complexidade aumenta quando ainda precisa-se manipular data-warehouses e sistemas web, nos quais várias *views* são diariamente definidas ou modificadas por usuários que desconhecem os detalhes dos metadados das fontes de dados e os seus inter-relacionamentos. É apresentada uma proposta com intuito de facilitar a definição das views (ou queries) a partir do esquema e algumas restrições de integridade. Um conjunto de queries em potencial é gerada que correspondem a view do usuário. Essa solução é baseada na existência de metadados descrevendo fontes individuais, semânticas descrevendo similaridades inter-fontes entre conceitos, e em algumas heurísticas que reduzem o tamanho do conjunto de queries.

Em outras palavras como definir queries a partir das views geradas das várias fontes de dados utilizando metadados para as fontes.

3.2.7 Heterogeneous Database Interoperability using the WWW

O uso de ferramentas da Internet pode ser uma maneira econômica, simples e independente de plataforma e SGBD para a integração de informações de bases de dados heterogêneas.

É proposta uma ferramenta para acesso integrado a essas bases heterogêneas através da internet. Essa solução é baseada em conceitos de mapeamento já conhecidos e pelo uso de tecnologias em internet. Portanto, o foco é em como integrar os dados de diversas fontes utilizando a internet.

3.2.8 MetaSIG: Ambiente de Metadados para Aplicações de Sistemas de Informações Geográficos

A tecnologia de SIG envolve diversos ramos e áreas de conhecimento, incluindo Banco de Dados. Especificadamente os BDs em SIG são conhecidos como "Banco de Dados Geográficos" que é um dos principais componentes do SIG uma vez que é o responsável pelo armazenamento e gerenciamento de todos os dados geográficos.

Os dados convencionais e gráficos dos dados geográficos muitas vezes precisam ser mostrados de forma diferenciada. Para tanto, tem sido propostas extensões de modelos concentuais tradicionais através da adição de primitivas geográficas com semântica espacial. Modelos Conceituais Geográficos são os modelos que possuem essa extensão.

Devido a uma diversidade de áreas de aplicação de SIG, estes geralmente se especializam em operações vetoriais (muito usada para aplicações urbanas) ou em operações matriciais (mais para aplicações ambientais).

Esta diversidade de modelos conceituais geográficos não é interessante para que se obtenha uma padronização e interoperabilidade. O MetaSIG se propõe a ser "genérico", ou seja, ser capaz de extrair metadados de aplicações de SIG e apresentá-los sem estar atrelado a um modelo específico. Para esta finalidade, foi usada a "arquitetura de quatro níveis" para metamodelagem [RAT99]. Conforme o trabalho original as camadas podem ser descritas assim:

- "Dados do Usuário": refere-se a um domínio de informação específico [RAT99]. Dentro do escopo de SIG, poderíamos considerar como instâncias desta camada os dados de aplicações geográficas armazenados em Bancos de Dados Geográficos de ambientes de SIG, como o MGE e o ARC/Info.

- "Esquemas": contém os esquemas conceituais resultantes da aplicação de um modelo, como o E-R, sobre um determinado domínio de informação.

- “Meta-esquemas”: contém os esquemas conceituais cujos domínios de informação. Referem-se, especificamente, aos modelo de dados, sejam eles modelos conceituais ou físicos, como por exemplo, o modelo físico do ambiente de SIG MGE e o Modelo Conceitual Geográfico Geo-OMT [BOR97].

- “Metaesquemas”: pode ser considerada como sendo o “Nível dos Modelos”.

- “Metameta-esquema”: define a linguagem para especificar metaesquemas. Ou seja, através de um determinado metameta-esquema deve ser possível criar instâncias de meta-esquemas, como por exemplo, os meta-esquemas dos Modelos Conceituais Geográficos.

3.3 XMLS+Matcher

Foi apresentado um modelo gráfico para representar os esquemas das fontes locais e o esquema do mediador definidos em XML Schema, o que foi chamado de XML Schema Semântico (XMLS).

Como já abordado, *matching* de esquemas é o processo de análise dos esquemas para identificação dos elementos correspondentes.

Foi proposto um método de matching chamado de XMLS+Matcher. Este método é baseado numa definição formal dos diversos tipos de assertivas de correspondência que são usadas na especificação formal do relacionamento semântico entre elementos de esquemas XMLS+.

Uma característica interessante desta proposta é neste modo de se especificar as correspondências entre o esquema do mediador e os esquemas locais. O XMLS+Matcher é ainda baseado em heurísticas e regras de inferência que auxiliam na identificação das correspondências [FIG97]. As informações oriundas dos esquemas são utilizadas pelas heurísticas além das informações provenientes de consultas a dicionários semânticos.

3.4 INTEGRA

INTEGRA é o sistema de integração de dados proposto por Bernadette Farias [SEM03]. Ele propôs a definição de uma arquitetura para sistemas de integração de dados e a definição de um processo com o objetivo de fazer a atualização dos sistemas de mediação, citando as principais contribuições do trabalho proposto.

A arquitetura é subdividida em quatro ambientes:

- *Common Core:* este ambiente é diretamente relacionado com *Data Integration Space* e com o *Mediator Generation and Maintenance Space*. Nas bases de dados haverá todos os dados colocados à disponibilidade do sistema de integração. Em geral aceitam diversos tipos de bases de dados presentes, o que é caracterizado como heterogeneidade. Além disso são capazes de funcionar sem depender do sistema de integração usado e podem sofrer modificações constantes do sistema, e por isso são chamadas de dinâmicas.

- *Data Integration Space:* este módulo é responsável por tratar aspectos que dizem respeito à estrutura de decomposição da consulta dos dados. Este ambiente possui dois módulos essenciais: o Query Manager e o Source;

- Manager. A principal tarefa desses módulos é de decompor as consultas e fazer a integração dos dados.

- Mediator Generation and Maintenance Space: o objetivo principal é gerar e realizar as manutenções das consultas de mediação. Este ambiente possui alguns módulos importantes para sua composição:

    - Conceptual Schema Manager: está relacionado à atualização do esquema de mediação, fazendo com que eventos sejam gerados para que a atualização seja realizada.

    - Schema Matcher: identifica possíveis relações entre os elementos do esquema a partir de uma análise sintática e semântica. É possível a relação lógica entre elementos de esquemas distintos.

- Mediation Queries Generator: realiza a identificação de possíveis operadores que podem ser usados entre os elementos das fontes de dados e também selecionando as fontes de dados que são encontradas.

- Mediation Queries Maintainer: este módulo possui a responsabilidade de tratar inclusão, alteração e a exclusão de bases de dados que foram ligadas ao sistema de integração.

- Mediation Queries Quality Evaluator: realiza uma análise do impacto de modificações no esquema da qualidade global do sistema, utilizando como métricas a disponibilidade da base de dados e o tempo de resposta que são utilizados dentro deste módulo.

- User Space: é o ambiente onde as questões referentes às especificações do usuário são definidas. Quando existe algum tipo de solicitação do usuário, como uma consulta, o sistema realiza uma avaliação dos dados disponíveis na base e o sistema deverá realizar a resposta à consulta.

- Lookup[3]: é responsável por capturar o esquema das bases de dados locais. Esses dados que são recuperados e relativos ao esquema da base são enviados a outro módulo do sistema. Este módulo que é responsável por comparar o esquema e também por gerar eventos de atualização do esquema de mediação.

3.5 CONCLUSÃO

Foram apresentados diversos artigos, pesquisas, documentos e projetos com finalidades semelhantes, e observa-se que nenhum apresenta o foco da proposta deste trabalho.

Algumas das abordagens citadas anteriormente somente se propoem a oferecer alguma infra-estrutura tecnológica para a integração. Por exemplo, tecnologias de comunicação (CORBA,DCOM) ou sistemas gerenciadores de banco

de dados (Ozone). Esses projetos por si só não são capazes de realizar a integração completa e transparente. São de grande valia como ferramentas no intuito de alcançar esse objetivo, mas não auto-suficientes.

Existem também trabalhos mais focados na manutenção do esquema integrado, sem especificar como o esquema seria gerado.

Outras abordagens possuem a proposta de realizar a integração dos dados usando XML. Geralmente sugerem algum modelo conceitual de como realizar essa integração, mas sem fornecer ferramentas práticas necessárias para esta finalidade.

Por fim os trabalhos que indicam algumas ferramentas apresentam características muito fechadas que podem ser difíceis de serem extendidas. Há casos em que essas ferramentas são para integrações de complexidade mais baixa, o que nem sempre corresponde à realidade.

O presente trabalho além de fornecer toda uma base teórica disponibiliza uma *API* capaz de gerar todos os *schemas* XML necessários. Desta forma, é possível se efetuar a integração dos dados.

A API serve também como uma poderosa ferramenta aos desenvolvedores de aplicações e administradores de dados, pois fornece visualizações das informações de maneira integrada. Essa transparência para visualização dos dados, sem haver necessidade de preocupação em termos de localização das fontes, é de grande valia nos projetos de sistemas. Além disso, por se tratar de uma API, é possível extendê-la e adicionar ou modificar funcionalidades de maneira menos complexa e mais rápida.

4 XML

4.1 INTRODUÇÃO

A sigla **XML** corresponde a Linguagem de Marcação Expansível (*eXtensible Markup Languag*e), e consiste em um padrão utilizado para a marcação de documentos que contêm informações estruturadas, ou seja, documentos que contêm uma estrutura clara e precisa da informação que é armazenada em seu conteúdo.

XML possibilita uma melhor estruturação da informação que permite a definição de documentos onde claramente são separados: conteúdo, significado e apresentação. Com algumas vantagens importantes, como, por exemplo, um mesmo conteúdo poderá ser apresentado em diversos formatos diferentes. Dessa forma, os documentos disponibilizados em XML poderão ser indexados de forma mais precisa que as tradicionais páginas planas escritas em HTML.

XML é um padrão para troca de informações entre diversas plataformas, que apenas possibilita a descrição de dados, em um arquivo de formato texto. Várias "*linguagen*s" derivadas de aplicações que utilizam os recursos de XML a torna uma poderosa ferramenta para a publicação de informações. A linguagem XML é um formato normalizado definido pelo World Wide Web Consortion (W3C) para a troca de qualquer tipo de informação.

É uma linguagem para descrever informação estruturada, independente de qualquer aplicação, sistema operativo ou base de dados, portanto totalmente aberta sem depender de qualquer tecnologia, empresa ou geografia. De fato, em fevereiro de 1998, a especificação XML tornou-se uma Technical Recommendation do W3C e tem o suporte de todas as empresas conhecidas de informática.

4.2 SINTAXE DE XML

Um dos fatores que torna XML uma linguagem bastante usada e difundida é a sua simplicidade. Pode-se encontrar basicamente dois tipos de estruturas em um documento XML: elementos e atributos.

Elementos são considerados a principal estrutura de um arquivo XML. O conteúdo de um elemento pode ser composto de: outros elementos, caracteres ou outros elementos e caracteres. Os elementos são delimitados por marcadores tais como <autor> e <autor> apresentado na figura 4.1. Além disso, é importante que cada *tag* de início (<livro>) tenha uma *tag* final correspondente. Também pode haver documentos XML vazios. A figura 4.1 mostra que *tag* <editora> está vazia nas suas duas ocorrências. Na primeira observa-se <editora/> e na segunda <editora></editora>.

```
<?XML version="1.0" encoding="utf-8"?>
<livraria>
  <livro id="L01" ano="1997">
    <autor>
      <nome>Marie</nome >
      <sobrenome>Buretta</sobrenome >
    </autor>
    <titulo>Data Replication</titulo>
    <editora/>
  </livro>
  <livro id="L02" ano="1996" bib="L01">
    <autor>
       <nome>Serge </nome>
       <sobrenome>Abiteboul</sobrenome >
    </autor>
    <titulo>Foundations of Data Bases</titulo>
    <editora> </editora>
  </livro>
</livraria>
```

Figura 4.1 – Livraria

Atributos também podem ser usados em documentos XML. Eles normalmente definem propriedades dos elementos e adicionam novas características. Somente *tags* iniciais ou *tags* vazias podem conter atributos. Na figura 4.1 temos exemplos de atributos: na primeira *tag* <livro> temos os atributos id="L01" e ano="1997", e na segunda, temos id="L02", ano="1996" e bib="L01".

Não há regras claras que indiquem quando usar elementos ou atributos. Uma atividade bastante comum é de sempre colocar dados como elementos e informação sobre dados (metadados) em atributos. E quando houver dúvidas, usar elementos. Há ainda alguns detalhes que devem ser analisados: 1)atributos não podem ser estruturados e 2) elementos são mais extensíveis em caso de necessidade de mudanças.

Um mesmo atributo não pode ser repetido em um mesmo elemento, diferentemente do que acontece com elementos. A primeira linha de um documento XML é chamada de prólogo e inclui a declaração XML, a declaração do tipo (versão) do documento e ainda a definição do tipo de codificação que é usada no arquivo.

4.3 FUNCIONALIDADE

XML possui uma extensa relação de bibliotecas e APIs (*Application Programming Interface*) que as linguagens vêm oferecendo para manipular arquivos XML. Bibliotecas como DOM (*Document Object Model*), SAX (*Simple API for XML*) e outras vêm sendo cada vez mais aprimoradas para permitir leitura, manipulação e escrita mais eficientes de arquivos XML.

Estas linguagens oferecem maneiras diferentes de tratar o arquivo XML. O DOM focaliza na apresentação do documento como um conjunto de objetos, SAX é uma interface que apresenta um documento XML como uma seqüência de eventos. Há ainda diversas ferramentas que permitem que um documento XML seja analisado com intuito de verificar se está bem formatado e se é válido. No primeiro caso, é verificado se o documento está de acordo com as normas da especificação XML. No segundo caso, o XML é validado em função de uma gramática (ou esquema) que define sua estrutura. Atualmente, há duas propostas principais para definição de esquemas XML: DTD (*Document Type Definition*) e XML Schema, sendo este último mais aceito como padrão.

4.4 XML Schema

Há, basicamente duas linguagens para validação de documentos XML: DTD e XML *Schema*, apresentadas na seção anterior. Apenas XML Schema será discutida nas seções seguintes pois é objetivo do trabalho a representação através XML Schema da integração de bancos de dados

4.4.1 Introdução a XML Schema

XML *Schema* foi proposta pelo W3C em 1999 e é uma linguagem XML usada para descrever e criar restrições sobre o conteúdo de documentos XML. De uma forma mais abrangente, o propósito de XML *Schema* é definir e descrever uma classe de documentos XML que segue uma série de regras impostas em um arquivo xsd (extensão de um arquivo XML *Schema*). O termo “documento instância” é o nome dado a um XML que se adequar às regras de um esquema particular.

XML *Schema* usa tipos de dados pré-definidos e pode definir restrições complexas. Além disso, por XML *Schema* ser um documento XML, diversos novos produtos têm permitido maior compatibilidade com XML *Schema*. Isso implica que qualquer ferramenta usada para manipular arquivos XML pode manipular arquivos XML Schema. A *tag* padrão usada como raiz de documentos XML *Schema* é <schema>.

Para expressar cardinalidade, por exemplo, XML *Schema* usa-se minOccurs (indica a quantidade mínima do elemento) e maxOccurs (indica a quantidade máxima de elementos) na definição de cardinalidade.

Na figura 4.2 observa-se o documento veiculo.XML. Verifica-se que o elemento <veiculo> possui alguns sub-elementos <passageiro>.

```
<?XML version=”1.0”>
<veiculo>
        <passageiro> Ana Carolina Salgado </passageiro>
        <passageiro> Pablo Sampaio </passageiro>
        <passageiro> Juliana </passageiro>
</ veiculo >
```

Figura 4.2 – Arquivo Veiculo.XML

O elemento element é usado na definição de elementos em XML *Schema*. Dois atributos devem ser adicionados na declaração de um elemento para informar o nome e o tipo. Os atributos são *name* e *type*. A figura 4.3 mostra a especificação básica de um elemento. Observa-se que há ainda as opções que normalmente têm restrições que devem estar relacionadas a cada elemento.

```
< element      name=″[Nome do elemento]″
               type=″[Tipo do elemento]″
               [Opções...]
>
```

Figura 4.3 – Especificação Básica de um Elemento

Os elementos podem ser classificados em simples e complexos. Os simples têm a característica de possuírem somente texto. Os complexos podem ser dos seguintes tipos: 1) Elementos vazios; 2) Elementos que contêm apenas outros elementos; 3) Elementos que contêm apenas texto; 4) Elementos que contêm texto e outros elementos.

Cada um dos tipos de elementos acima pode possuir atributos. Um elemento que possui somente texto pode ser definido da seguinte maneira:

<element name=”empregado” type=”string”/>

Um elemento em XML que se adequar a esta declaração é:

<empregado>João da Silva </empregado>

Um esquema que define um elemento que possui apenas outros elementos pode ser exemplificado como na figura 4.4.

```
<element name=″empregado″ type=″string″>
   <sequence>
      <element name=″nome″ type=″string″>
      <element name=″idade″ type=″integer″>
   </sequence>
</element>
```

Figura 4.4 XML Schema: Empregado

XML *Schema* apresenta tipos de dados pré-definidos e permite que o usuário crie seus próprios tipos. Os tipos de dados em XML *Schema* podem ser divididos em quatro grupos: Texto, Datas, Numéricos e Miscelâneas.

No grupo *texto*, os principais tipos de dados são: string, token e normalizedString. A diferença entre estes três tipos é bastante sutil e refere-se somente a como o processador XML trata quebras de linhas, espaços em branco, tabulações, entre outros.

O grupo *Datas* contém alguns tipos de dados usados para armazenar informações de datas, horas, durações, entre outros. Os principais tipos de dados neste grupo são: date, dateTime, time e duration.

Há também o grupo de *numéricos* que define os tipos de dados responsáveis por tratar com números. Alguns dos exemplos de tipos de dados pertencentes a este grupo são: integer, byte, decimal, long, unsignedInt.

Por último, há o tipo de miscelâneas que define os tipos de dados restantes. Os principais são: boolean, hexBinary e anyURI.

XML *Schema* ainda permite que o usuário crie seus próprios tipos de dados e os utilize onde for conveniente. O elemento <complexType> é usado para este fim. A figura 4.5 uma simples especificação de <complexType>.

```
<complexType name=”[Nome do tipo]”>
        <[Especificação do elemento]>
        <[Especificação do elemento]>
        <[Especificação do elemento]>
        ....
</complexType>
```

Figura 4.5 - Especificação de tipos complexos

Há uma outra categoria de tipos complexos: são os chamados tipos de dados implícitos. Com o uso destes, é permitido que seja definida uma *tag* <element> sem o atributo type. Substitui-se este atributo por uma definição de tipo complexo interior a *tag* <element>. A figura 4.6 na página seguinte mostra como um tipo de dados implícito.

```
<element name=”empregado” >
   <complexType>
      <sequence>
         <element name=”nome” type=”string”>
         <element name=”idade” type=”positiveInteger”>
         <element name=”dataNascimento” type=”dateTime”>
         <element name=”casado” type=”boolean”>
         <element name=”paginaPessoal” type=”anyURL”>
      </sequence>
   </complexType>
</element>
```

Figura 4.6 - Tipo de dado implícito

XML *Schema* define um elemento attribute que especifica que atributos são permitidos em um determinado elemento. A figura 4.7 mostra uma simples especificação de um attribute.

```
< attribute    name=”[Nome do atributo]”
               type=”[Tipo do atributo]”
               [Opções...]
>
```

```
< attribute    name=”[Nome do atributo]”
               type=”[Tipo do atributo]”
               [Opções...]
>
```

Figura 4.7 - Especificação de Atributos

A figura 4.8 apresenta como definir um atributo em um documento XML *Schema* e como esse atributo é refletido no documento XML.

```
<element name=”empregado” type=”informacoesPessoais”>

<complexType name=”informacoesPessoais”>
   <sequence>
      <element name=”nome” type=”string”>
      <element name=”idade” type=”positiveInteger”>
      <element name=”dataNascimento” type=”dateTime”>
      <element name=”casado” type=”boolean”>
      <element name=”paginaPessoal” type=”anyURL”>
   </sequence>
   <attribute name=”codigo” type=”integer”>
</ complexType>
```

Figura 4.8 - XML Com Atributo

A figura 4.8 apresenta uma parte da declaração de atributos onde alguns itens opcionais podem ser adicionados. Os mais comuns são use, default e fixed. O nome use pode receber os valores optional (indicando que a presença deste atributo é optativa) ou required (obrigando que o atributo esteja presente no elemento). Quando se usa o atributo default na declaração de um attribute significa que haverá um valor padrão para o atributo quando este não for informado. O atributo fixed indica que todos os atributos definidos deverão ter sempre o mesmo valor e não poderão ser modificados. Nas unicidades e chaves, XML *Schema* permite que os conceitos de unicidade e de chaves estejam presentes em um documento XML. Para isso, são usados os elementos unique e key.

1)key: pode ser aplicado a um elemento, um atributo ou uma combinação de elementos de atributos. O uso de key implica que o elemento ou atributo será único, não nulo e estará sempre presente.

2)unique: também pode ser aplicado a um elemento, um atributo ou uma combinação de elementos e atributos. O seu uso implica que o elemento ou atributo será único e não nulo. Ou seja, a única diferença entre key e unique está no fato de que key implica a presença do elemento ou do atributo. A figura 4.9 mostra a sintaxe básica de key.

```
<?XML version="1.0"?>
<xsd:schema XMLns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.books.org"
            XMLns="http://www.books.org"
            XMLns:bk="http://www.books.org"
            elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:elementname="Title" type="xsd:string"/>
                            <xsd:element name="Author"
type="xsd:string"/>
                            <xsd:element name="Date" type="xsd:string"/>
                            <xsd:element name="ISBN" type="xsd:string"/>
                            <xsd:element name="Publisher"
type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:key name="PK">
            <xsd:selector xpath="bk:Book"/>
            <xsd:field xpath="bk:ISBN"/>
        </xsd:key>
    </xsd:element>
</xsd:schema>
```

Figura 4.9 - Arquivo Bookstore.Xsd

Há dois elementos internos ao elemento key: selector e field. O primeiro é usado para identificar qual o elemento que contém a restrição. O segundo indica qual o campo do elemento que será usado para testar a unicidade, como apresenta a figura 4.10.

```
<xsd:unique name="PK">
        <xsd:selector xpath="bk:Book"/>
        <xsd:field xpath="bk:ISBN"/>
</xsd: unique >
```

Figura 4.10 - Unique

4.5 A LINGUAGEM XML COMO FERRAMENTA DE INTEGRAÇÃO

A necessidade de integrar os dados de negócios, tanto internamente quanto com parceiros comerciais, já é item prioritário nas agendas dos responsáveis pela área de Tecnologia da Informação. Mas para que essa visão seja concretizada, um elemento é vital: o metadado, a informação sobre a informação, que define localização, contexto, premissas e restrições de uso por trás dos dados corporativos. Em geral, ela encontra-se escondida, dispersa ou simplesmente não existe. Analistas concordam que a informação contextual relativa a projetos, registros e aplicativos raramente tem sido armazenada e disponibilizada de modo estruturado. Por isso, o gerenciamento do metadado está se tornando um ponto de pressão dentro das corporações, conforme suas fronteiras perdem definição em rede. Administrar o metadado não é uma preocupação nova, o que se deve lembrar é que "XML é uma linguagem do tipo mark-up, e não uma solução de gerenciamento de metadados. XML pode ajudar bastante, viabilizando a troca de informações entre as ferramentas, mas não oferece facilidades de administração."

O mercado que endereça essa questão está hoje dividido entre propostas de amplos repositórios empresariais; soluções do tipo framework em áreas específicas – como data warehousing ou systems management – e soluções baseadas em XML, que ele chama de coringas do mercado. Note-se que a tecnologia XML, além de ser um caminho alternativo, também é usada no suporte às duas outras soluções. Entender as novas necessidades de integração aos usuários, os analistas aconselham cautela: a nova geração de ferramentas para gerenciamento de metadados ainda é imatura, onde enfatizamos a proposta de integração com aplicação de dicionário em forma de api.

Muitas ferramentas já oferecem hoje facilidades restritas para gerenciar metadados, e a expansão dessas funcionalidades representa excelentes oportunidades para os fornecedores. Exemplos são as empresas de data warehousing, que cada vez mais destacam a circulação de metadados entre as ferramentas de extração, transferência e carregamento entre a warehouse e o software de OLAP (On-line Analytical Processing) [TRI1997].

4.5.1 Microsoft MSXML

MSXML é uma ferramenta adotada pela plataforma.net pela Microsoft, que foi criada com base na estrutura XML e XMLSchema, e foi evoluída conforme a necessidade de sua aplicação.

O XML Schema da Microsoft tem com vantagem principal de, ao selecionar as tabelas desejadas de um banco de dados, gera documentos no formato XML recuperando todas as colunas e os tipos de dados da tabela a que foi referenciado. Recupera também toda a integridade referencial do banco e como inovação, traz a informação de quais atributos das tabelas são “primary key”. Possui a vantagem de criar chaves primárias, estrangeiras e originais, todos esses são criados a partir de uma série de dados de um documento XML.

Os elementos e atributos continuam com a mesma estrutura que foi utilizada pela W3C, a final o MSXML é um complemento à estrutura inicial da XML Schema.

O MSXML foi criado com o propósito de gerar documentos que complementasse as necessidades do usuário. A XML é usada para muitos fins, mas apesar de não poder ser monopolizada por empresas, existe a possibilidade de criação de evoluções específicas, estudos criados a partir de um padrão XML criado e que não pode ser alterado, mas sim pode ser ajustado.

Exemplo de XML Schema Microsoft

```
xs:element name="NewDataSet" msdata:IsDataSet="true">
 <xs:complexType>
  <xs:choice maxOccurs="unbounded">
   <xs:element name="Customers">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="CustomerID" type="xs:string"/>
      <xs:element name="CompanyName" type="xs:string"/>
      <xs:element name="ContactName" type="xs:string"
minOccurs="0"/>
      <xs:element name="ContactTitle" type="xs:string"
minOccurs="0"/>
      <xs:element name="Address" type="xs:string" minOccurs="0"/>
      <xs:element name="City" type="xs:string" minOccurs="0"/>
      <xs:element name="Region" type="xs:string" minOccurs="0"/>
      <xs:element name="PostalCode" type="xs:string"
minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:choice>
</xs:complexType>
  <xs:unique name="NewDataSetKey1" msdata:PrimaryKey="true">
```

4.6 CONCLUSÃO

O avanço tecnológico dentro de aplicações de Business to Business necessita principalmente de dados confiáveis e com a capacidade de adaptabilidade a qualquer tecnologia em que seja aplicado.

O documento gerado com um formato XML possibilita ao usuário que seus dados sejam flexíveis a mudanças. XML foi criado a partir da necessidade que existia de que, com o crescimento de aplicações, que interagem mesmo com bancos de dados e plataformas diferentes, que precisavam utilizar os mesmos dados de forma íntegra e rápida.

A partir desse padrão de documentos XML, muitas ferramentas podem ser geradas, para serem utilizados nos mais variados tipos de aplicações, no caso desse estudo, a ferramenta proposta é voltada para a manipulação dos dados, diretamente ligada ao banco de dados, considerando data types podendo ser validados em qualquer aplicação ou plataforma. A transição dos dados deve ser garantida com segurança, e para isso ferramentas de aplicações específicas para “conversação” dessas plataformas e/ou aplicações seja facilitada sem limitações.

5 PROPOSTA DE INTEGRAÇÃO

5.1 INTRODUÇÃO

Apresenta-se como proposta de integração de MultiDatabases heterogêneos com aplicação de XML Schemas.

5.2 ARQUITETURA DE INTEGRAÇÃO DE MULTIDATABASES HETEROGÊNEOS

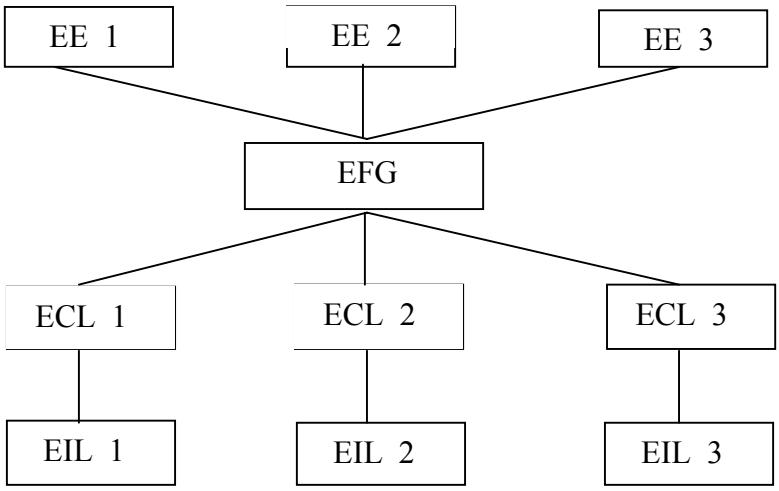


Figura 5.2 – Proposta da Arquitetura de Integração

A arquitetura de integração representada na figura 5.1 é uma extensão da arquitetura ANSI/SPARC para multidatabases sem esquema global. Esta proposta de arquitetura de integração de multidatabases heterogêneos consiste nas descrições das fontes locais que chamamos de esquemas conceituais locais (ECL), onde são interpretadas as semânticas dos esquemas internos locais (EIL). Correspondências entre os esquemas conceituais das fontes, localização e métodos de acesso aos SGDBs componentes do sistema multidatabase compõem o esquema funcional global (EFG), sendo um núcleo de funcionalidades, que a diferencia das demais arquiteturas. E, finalmente a disponibilização para o usuário final de um esquema externo (EE).

Uma das razões que levou a utilização desta arquitetura foi à necessidade de obter o conhecimento dos esquemas conceituais locais de forma dinâmica e

independente da localização dos SGDBs componentes.

5.3 ESTRUTURA DE INTEGRAÇÃO DE MULTIDATABASES HETEROGÊNEOS

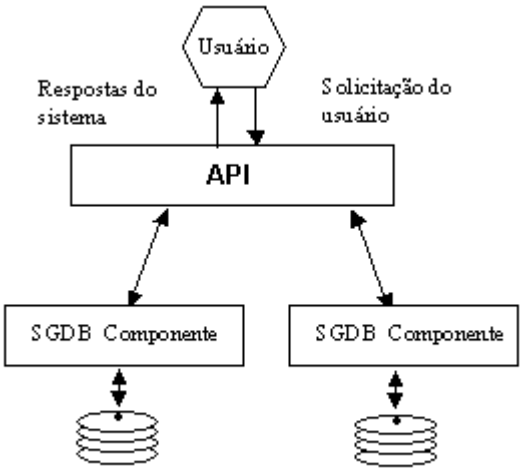


Figura 5.2 - Estrutura de Integração de Multidatabases Heterogêneos

A integração de Multidatabases com a utilização de uma API representada na figura 5.2 fornece uma camada de software que funciona sobre esses SGBDs individuais e proporciona aos usuários os recursos de acesso a diversos bancos de dados bem como o conhecimento sobre os metadados desses SGDBs individuais. Descrevemos a seguir os componentes desta estrutura:

Usuário

O responsável pela identificação do esquema local dentro do ambiente é também chamado de "*broker*". Este usuário, além de possuir conhecimentos sobre modelagem de dados, deve conhecer o modelo das fontes, para decidir quais metadados farão parte do dicionário. Estes metadados podem ser automaticamente extraídos da fonte ou ainda podem ser referenciados de um modelo previamente definido usuário.

API

É uma camada de software que manipula todas as funcionalidades que estarão atuando sobre os SGDBs componentes. As funcionalidades são: manipulação do dicionário de metadados que contém os ECLs, acesso aos SGDBs componentes pelos drivers nativos para maior rapidez na recuperação dos dados,

tratamento dos ECLs para integração dos SGBDs componentes e geração dos EEs.

SGDBs componentes

As Fontes de Informação são heterogêneas, com modelos de dados e linguagens de consultas próprias e autônomas. Além disso, elas podem suportar aplicações locais e atualizar suas informações independentemente, sem se preocupar se estas atualizações afetarão o sistema de integração de informações que as mesmas participam.

Não será apresentado aqui todo o código fonte por inteiro, uma vez que para esse fim seria recomendada a disponibilização completa do software com seu código e a possibilidade de visualização na IDE específica, no caso o Microsoft Visual Basic.

1. A função *geraDicionario()*

```
Function geraDicionario()
i = 0
j = 0
   While i < 20
      While j < 100000
         gmatriz_Dicionario(i, j) = "" ‘INICIALIZA-SE A MATRIZ
         j = j + 1
      Wend
      i = i + 1
   Wend
‘ADICIONA-SE À MATRIZ OS ITENS (atributos, tabelas, etc)
   gmatriz_Dicionario(0, 0) = " bortoms"
   gmatriz_Dicionario(0, 1) = "roapi"
   gmatriz_Dicionario(1, 0) = " bortoms.tb_pessoa"
   gmatriz_Dicionario(1, 1) = " bortoms.tb_endereco"
   gmatriz_Dicionario(1, 2) = " bortoms.tb_complemento_pessoas"
   gmatriz_Dicionario(1, 3) = " bortoms.tb_telefone"
   gmatriz_Dicionario(1, 4) = "roapi.tb_veiculos"
   gmatriz_Dicionario(1, 5) = "roapi.tb_marca"
   gmatriz_Dicionario(1, 6) = "roapi.tb_modelo"
```

2. A função *geraXMLSDicionario()*

```
Public Function geraXMLSDicionario()
    Dim strArquivo As TextStream
    'ABRIMOS O ARQUIVO
   Set strArquivo = fso.OpenTextFile("C:/Mestrado_API/XMLSG.XML", ForWriting)

'GRAVAMOS O XML COM OS DADOS REFERENTES AS BASES
strArquivo.WriteLine ("<schema XMLns='http://www.w3.org/2000/10/XMLSchema'
elementFormDefault='qualified'>")
    strArquivo.WriteLine ("<element name='RH' type='pub:tRH' />")
    strArquivo.WriteLine ("<complexType name='tRH'>")
    strArquivo.WriteLine ("<unique name='UnicidadeCodigoPessoa'>")
    strArquivo.WriteLine ("<selector>RH/TB_PESSOA/NM_PESSOA</selector>")
    strArquivo.WriteLine ("<field>@CD_PESSOA</field>")
    strArquivo.WriteLine ("</unique>")
    strArquivo.WriteLine ("<unique name='UnicidadeCodigoVeiculo'>")
    strArquivo.WriteLine ("<selector>VEICULOS/TB_VEICULOS/DS_DESCRICAO</selector>")
    strArquivo.WriteLine ("<field>@CD_VEICULO</field>")
    strArquivo.WriteLine ("</unique>")
    strArquivo.WriteLine ("<sequence>")
    strArquivo.WriteLine ("<element name='TB_PESSOA' type='pub:tPessoa' minOccurs='1'
maxOccurs='unbounded' />")
    strArquivo.WriteLine ("</sequence>")
    strArquivo.WriteLine ("<sequence>")
    strArquivo.WriteLine ("<element name='TB_VEICULOS' type='pub:tVeiculos' minOccurs='1'
maxOccurs='1' />")
    strArquivo.WriteLine ("</sequence>")
    strArquivo.WriteLine ("</complexType>")
    strArquivo.WriteLine ("<complexType name='tTB_Pessoa'>")
    strArquivo.WriteLine ("<sequence>")
    strArquivo.WriteLine ("<element name='CD_PESSOA' type='integer' />")
    strArquivo.WriteLine ("<element name='CD_CODIGO' type='pub:tTB_Endereco' minOccurs='1'
maxOccurs='unbounded' />")
    strArquivo.WriteLine ("<element name='CD_CODIGOTEL' type='pub:tTB_Tlefone' minOccurs='1'
maxOccurs='unbounded' />")
    strArquivo.WriteLine ("<element name='NM_PESSOA' type='string' />")
    strArquivo.WriteLine ("</sequence>")
    strArquivo.WriteLine ("</complexType>")
    strArquivo.WriteLine ("<complexType name='tTB_Endereco'>")
    strArquivo.WriteLine ("<complexContent>")
    strArquivo.WriteLine ("<extension base='pub:tTB_Pessoa'>")
    .
    .
    .
```

3. A função *gerarXMLSIndependente()*

```
Function gerarXMLSIndependente()

   Dim strArquivoSI As TextStream

   Set strArquivoSI = fso.OpenTextFile("XMLSI.XML", ForWriting)

   strArquivoSI.WriteLine ("<schema
XMLns='http://www.w3.org/2000/10/XMLSchema'
elementFormDefault='qualified'>")
   strArquivoSI.WriteLine ("<element name='RH' type='pub:tRH'/>")
   strArquivoSI.WriteLine ("<complexType name='tRH'>")

   strArquivoSI.WriteLine ("<unique name='UnicidadeCodigoPessoa'>")
   strArquivoSI.WriteLine ("<selector>RH/TB_PESSOA/NM_PESSOA</selector>")
   strArquivoSI.WriteLine ("<field>@CD_PESSOA</field>")
   strArquivoSI.WriteLine ("</unique>")

   strArquivoSI.WriteLine ("<unique name='UnicidadeCodigoVeiculo'>")
   strArquivoSI.WriteLine
("<selector>VEICULOS/TB_VEICULOS/DS_DESCRICAO</selector>")
   strArquivoSI.WriteLine ("<field>@CD_VEICULO</field>")
   strArquivoSI.WriteLine ("</unique>")

   strArquivoSI.WriteLine ("<sequence>")
   strArquivoSI.WriteLine ("<element name='TB_PESSOA' type='pub:tPessoa'
minOccurs='1' maxOccurs='unbounded'/>")
   strArquivoSI.WriteLine ("</sequence>")

   strArquivoSI.WriteLine ("<sequence>")
   strArquivoSI.WriteLine ("<element name='TB_COMPLEMENTO_PESSOA'
type='pub:tComplementoPessoa' minOccurs='1' maxOccurs='1'/>")
   strArquivoSI.WriteLine ("</sequence>")

   strArquivoSI.WriteLine ("<sequence>")
   strArquivoSI.WriteLine ("<element name='TB_VEICULOS' type='pub:tVeiculos'
minOccurs='1' maxOccurs='1'/>")
   strArquivoSI.WriteLine ("</sequence>")
   strArquivoSI.WriteLine ("</complexType>")

   strArquivoSI.WriteLine ("<complexType name='tTB_Pessoa'>")
   strArquivoSI.WriteLine ("<sequence>")
   strArquivoSI.WriteLine ("<element name='NM_PESSOA' type='string'/>")
   strArquivoSI.WriteLine ("</sequence>")
   strArquivoSI.WriteLine ("</complexType>")
```

4. O evento do click do botão PROCESSAR

```
'O codigo do evento ao clicar
'no botao PROCESSAR
Private Sub Command1_Click()

'instancia-se um objeto referente a API
Set obj = New ApiDll.clsTesteApi

'Passamos ao metodo criaAPI os seguintes dados, nesta ordem:
'Parametro1: o Banco de Dados
'Parametro2: a base de dados
'Parametro3: As tabelas
'Parametro4: os atributos das tabelas
ret = obj.criaApi(Combo1.ItemData(Combo1.ListIndex), Text1.Text, Text2.Text,
Text3.Text, Text4.Text)

If ret = -1 Then
   MsgBox "ERRO" 'Nao ocorreu o processamento
Else
   MsgBox "Processamento OK!" 'Processamento aconteceu corretamente
End If

End Sub
```

Nas próximas páginas serão apresentados os códigos do *front end* da aplicação. Assim é possível visualizar como ocorre a comunicação do usuário com a API, demonstrado uma forma de usá-la eficientemente.

5. O evento do click do botão GERAR XML

```
'O evento do click do botao GERAR XML
Private Sub Command2_Click()

'cria-se a instância referente a API
Set obj = New ApiDll.clsTesteApi
'Chamada ao método que gera o arquivo XML
ret = obj.GeraXML()

If ret = 1 Then
   MsgBox "Geração Finalizada" 'XML criado corretamente
Else
   MsgBox "ERRO"
End If

End Sub
```

O acesso à API foi projetado de tal modo que pareça transparente ao usuário. Foram definidos alguns métodos de entendimento indutivo pelo próprio nome (geraXML() por exemplo) que escondem toda a complexidade de quem estiver se utilizando da API.

5.4 ARQUITETURA DO ESQUEMA FUNCIONAL GLOBAL (EFG)

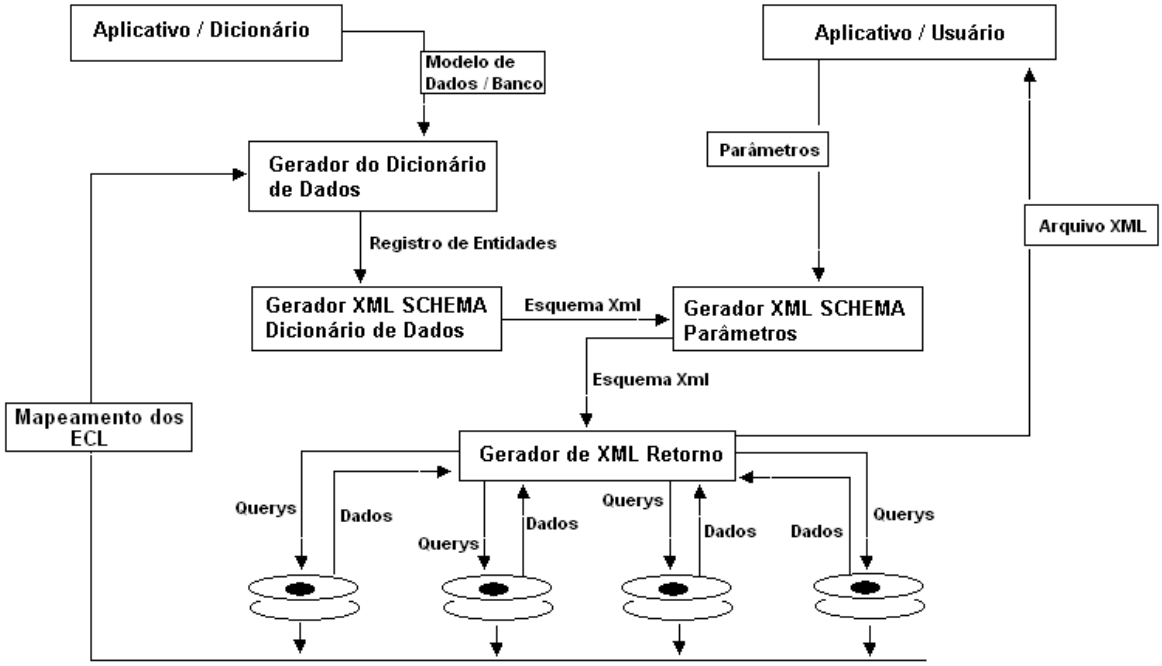


Figura 5.3 – Arquitetura do Esquema Funcional Global (EFG)

Descreve-se nas seções seguintes os componentes da arquitetura do EFG.

5.4.1 Mapeamento dos Esquemas Locais

Os esquemas locais de cada SGDBs componentes são mapeados, ou seja, é nesta etapa que identifica-se os ECLs e referencia-se as entidades encontradas com seus respectivos atributos, datatypes, chaves, relacionamentos e a identificação do SGDB componente. Esta coleção de ECLs permite a construção do XML Schema do dicionário com os mapeamentos efetuados.

5.4.2 Gerador do dicionário de dados

Tem por finalidade gerar o dicionário com as descrições das fontes de informação que participam do sistema de integração. O usuário terá como instâncias os ECLs dos SGDBs componentes tais como os esquemas locais ilustrados nas Figuras 5.5 e 5.6. O objetivo de manipular os esquemas na API é tornar dinâmica a

utilização pelos responsáveis da interface com o usuário. O EFG possibilita quais metadados devem ser extraídos de seus respectivos ECLs, através de determinados atributos que definem como esta extração será conduzida. O mesmo ocorre com o módulo de Interface com o Usuário que utiliza o EFG para determinar quais metadados devem ser apresentados para o usuário/dba.

Para criar a API que manipulará este dicionário foi utilizada a ferramenta de desenvolvimento Visual Basic versão 6 pertencente a Microsoft.

5.4.3 Dicionário

O conceito predominante para o suporte à administração da informação é o armazenamento de metadados, ou seja, uma tecnologia capaz de tratar as informações relativas aos dados (metadados), inclusive aquelas relacionadas ao contexto em que eles são utilizados. A manipulação de metadados pela API, possibilita a visualização de todo o ambiente de ECLs a serem integrados através dos inter-relacionamentos existentes e as associações com os processos funcionais, fluxos de informações, infra-estrutura de processamento e comunicação que será usada dinamicamente pelas aplicações. Desta maneira permite-se que os metadados possam ser incrementados a partir dos ECLs dos SGBDs e acessados pelas linguagens de programação.

O armazenamento dos ECLs de forma dinâmica tornariam mais confiáveis os ECLs apresentados. Entretanto, os seguintes motivos levaram a decidir pelo armazenamento dos ECLs mapeados:

a) a extração dinâmica certamente prejudicaria o desempenho dos aplicativos;

b) de acordo com usuários projetistas, as aplicações de alterações são executadas de forma planejada;

c) futuras alterações de projeto poderiam ser refletidas no dicionário de ECLs através de um mecanismo de controle de atualizações, que será desenvolvido em trabalhos futuros.

É importante ressaltar que a interface do Ambiente do dicionário apresentará os ECLs que se encontram armazenados no dicionário.

Um dos objetivos da utilização deste dicionário é permitir ao usuário final fazer consultas através de uma interface muito mais amigável, onde as perguntas a serem feitas não necessitariam seguir uma determinada linguagem de manipulação de dados.

5.4.4 Estrutura do Dicionário

Para a construção do dicionário, foi necessário o mapeamento dos ECLs levantados pelo usuário. A estrutura do dicionário é apresentada na figura 5.4., e as descrições são listadas a seguir:

- **Banco de dados:** refere-se ao nome do banco de dados levantados pelo usuário/dba. Possui a seguinte definição:
  - **Entidade:** refere-se as tabelas a serem extraídas
  - **Atributo:** refere-se as propriedades nome, tipo e chave
  - **Entidade / Relacionamento** refere-se à participação de uma determinada entidade em um relacionamento em particular.

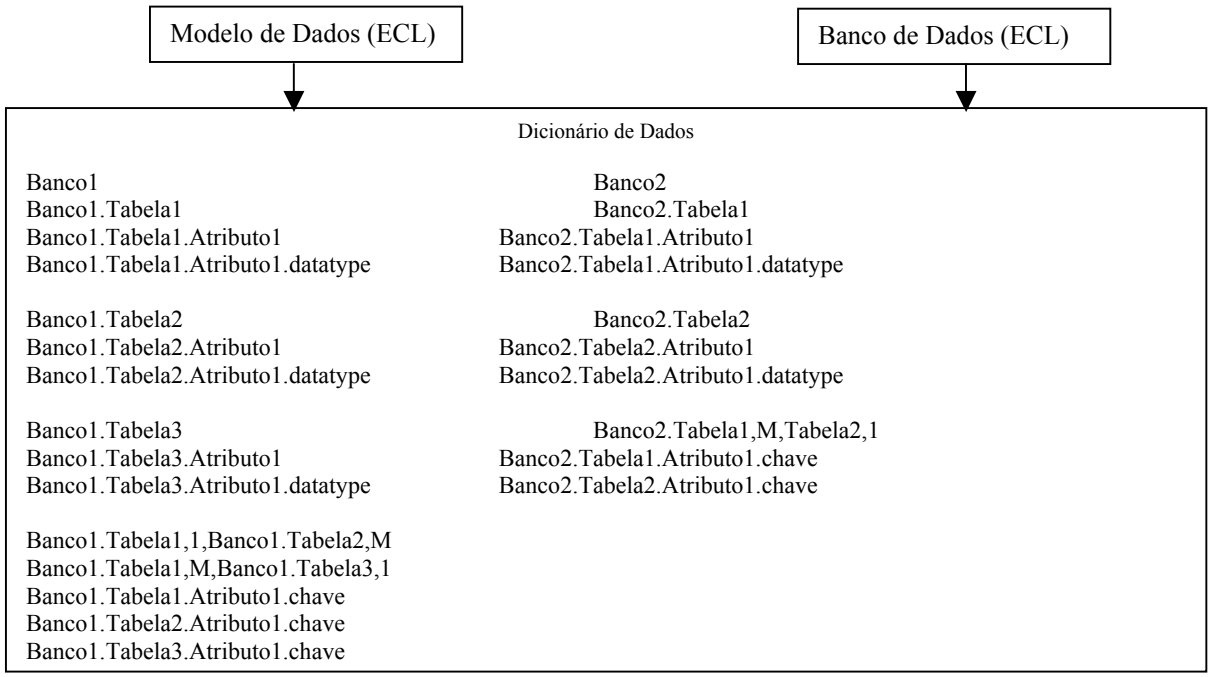


Figura 5.4 – Estrutura do Dicionário

Apresenta-se um exemplo com dois bancos de dados (a serem integrados sendo o primeiro banco de dados de RH (recursos humanos) onde o modelo de dados é representado na figura 5.5 e o segundo banco de dados de veículos representado no modelo da figura 5.6.

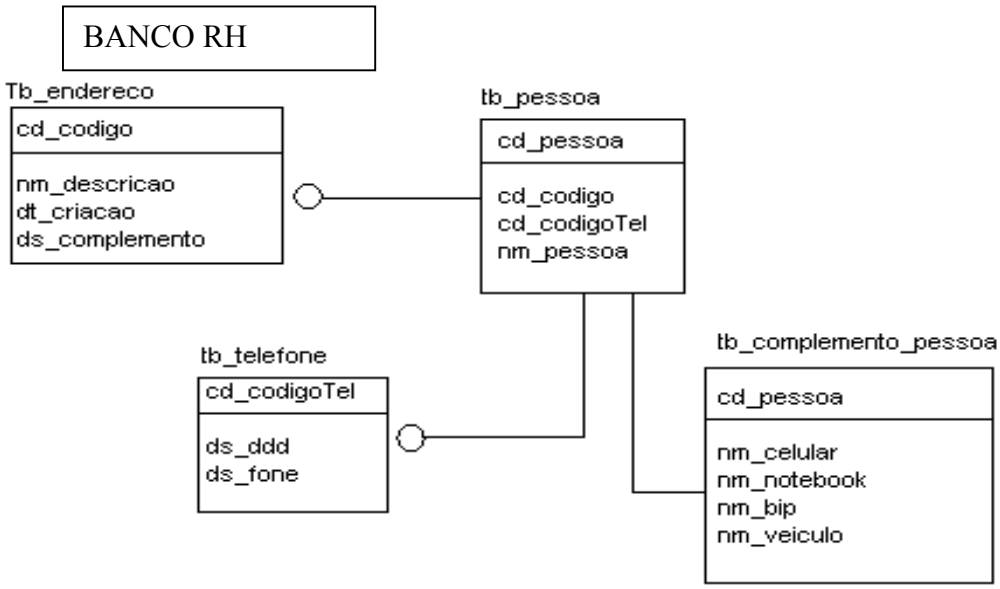


Figura 5.5 - Modelo do Banco de Dados RH

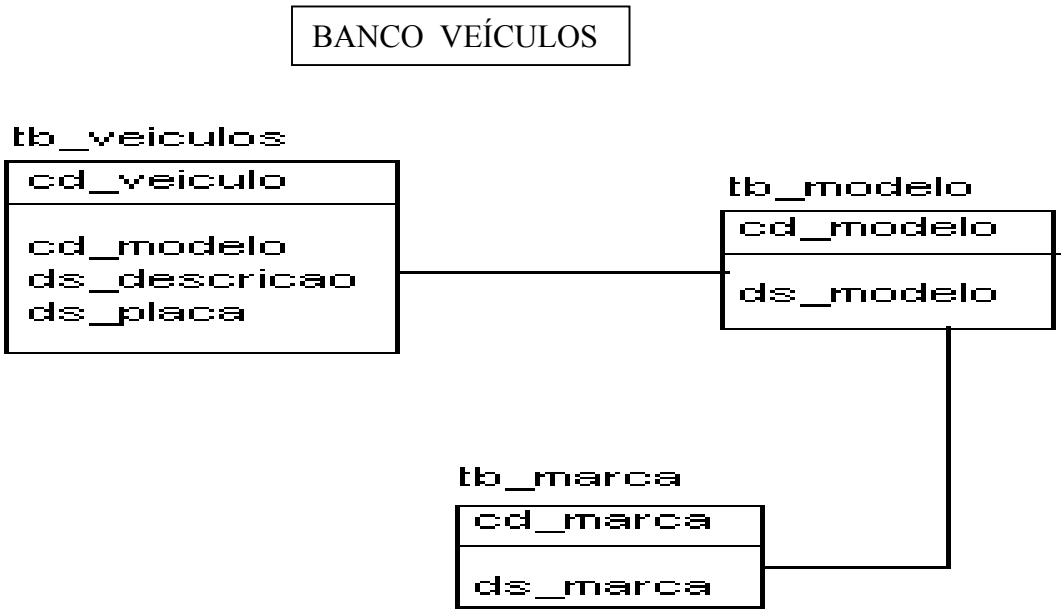


Figura 5.6 - Modelo do Banco de Dados Veículos

Define-se alguns passos para definição do dicionário a partir dos modelos apresentados:

1º passo: definição dos ECLs locais com identificação do banco de dados, entidades, atributos, datatypes, chaves, relacionamentos e conexão de acesso.

2º passo: inserir no dicionário uma estrutura que separe de forma hierárquica os níveis de identificação apresentados na figura 5.7.

O resultado da integração dos modelos apresentados na figura 5.5 e figura 5.6 ficara representado em forma de modelo conforme é apresentado na figura 5.7.

RH	→ BANCO DE DADOS RH
RH.TB_ENDERECO.NM_DESCRICAO.VARCHAR(100)	→ BANCO + TABELA + ATRIBUTO + DATATYPE
RH.TB_ENDERECO.DT_CRIACAO.DATE	
RH.TB_ENDERECO.DS_COMPLEMENTO.VARCHARF(50)	
RH.TB_TELEFONE.CD_CODIGOTEL.INTEGER	
RH.TB_TELEFONE.DS_DDD.INTEGER	
RH.TB_TELEFONE.DS_FONE.INTEGER	
RH.TB_PESSOA.CD_PESSOA.INTEGER	
RH.TB_PESSOA.CD_CODIGO.INTEGER	
RH.TB_PESSOA.CD_CODIGOTEL.INTEGER	
RH.TB_PESSOA.NM_PESSOA.VARCHAR(40)	
VEICULOS	→ BANCO DE DADOS VEICULOS
VEICULOS.TB_VEICULOS.CD_VEICULO.INTEGER	→ BANCO + TABELA + ATRIBUTO + DATATYPE
VEICULOS.TB_VEICULOS.CD_MODELO.INTEGER	
VEICULOS.TB_VEICULOS.DS_DESCRICAO.VARCHAR(40)	
VEICULOS.TB_VEICULOS.DS_PLACA.VARCHAR(20)	
VEICULOS.TB_MODELO.CD_MODELO.INTEGER	
VEICULOS.TB_MODELO.DS_MODELO.VARCHAR(20)	
VEICULOS.TB_MARCAS.CD_MARCA.INTEGER	
VEICULOS.TB_MARCAS.DS_MARCA.VARCAHAR(20)	
RH.TB_PESSOA,1, RH.TB_ENDERECO,M	→ RELACIONAMENTO TB_PESSOA – TB_ENDERECO
RH.TB_PESSOA,1, RH.TB_TELEFONE, M	→ RELACIONAMENTO TB_PESSOA – TB_TELEFONE
VEICULOS.TB_VEICULOS, 1, VEICULOS.TB_MODELO, 1	→ RELACIONAMENTO TB_VEICULOS – TB_MODELO
VEICULOS.TB_VEICULOS, 1, VEICULOS.TB_MARCA, 1	→ RELACIONAMENTO TB_VEICULOS – TB_MARCA
RH.TB_PESSOA.CD_PESSOA	→ CHAVE TB_PESSOA
RH.TB_ENDERECO.CD_CODIGO	→ CHAVE TB_ENDERECO
RH.TB_TELEFONE.CD_CODIGOTEL	→ CHAVE TB_TELEFONE
VEICULOS.TB_VEICULOS.CD_VEICULO	→ CHAVE TB_VEICULOS
VEICULOS.TB_MARCA.CD_MARCA	→ CHAVE TB_MARCA
VEICULOS.TB_MODELO.CD_MODELO	→ CHAVE TB_MODELO

Figura 5.7 - Dicionário Resultante dos Bancos de Dados RH e Veículos

5.4.5 Ferramenta de geração do dicionário

Constrói-se uma ferramenta de apoio a API para geração do dicionário através dos ECLs locais mapeados pelo usuário/DBA ou de um arquivo de dicionário que será importado como ECL para o dicionário.

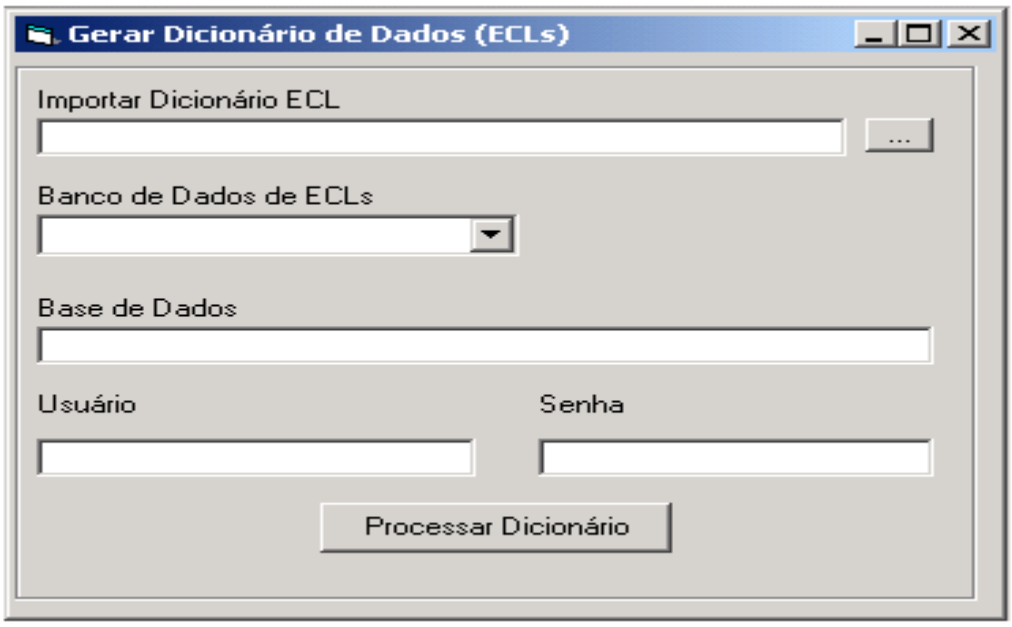


Figura 5.8 - Ferramenta de Geração do Dicionário

Verifica-se na figura 5.8 que a tela contêm cinco campos de entrada de dados para o processo de geração do dicionário, descritos a seguir:

1º campo - Importar Dicionário: informa a API a localização do arquivo do ECL para processar as informações necessárias à criação do dicionário de dados.

2º campo - Banco de Dados: informa qual o SGDB a API deve utilizar para realizar a conexão, que deve estar definida para os SGDBs componentes da integração.

3º campo - Base de Dados: informa a qual a base de dados específica do SGDB que a API deve utilizar para recuperar os metadados necessários a criação do dicionário.

4º campo - usuário: informa usuário que efetuará a conexão com o SGDB e base de dados informados anteriormente, necessariamente tendo permissão para esta operação.

5º campo - senha: informa senha do usuário.

Há três possibilidades de efetuar a criação do dicionário:

1) Informar o caminho do arquivo do ECL com o seguinte formato: *Banco de Dados, tabela, tabela.atributo, tabela.atributo.datatype, tabela.chave, tabela.relacionamento*.

2) Informar qual o SGDB a ser utilizado, o nome da base de dados, o usuário desta base de dados e sua senha respectivamente.

3) Utilizar tanto as duas possibilidades anteriores onde a API gerará simultaneamente os dois esquemas locais.

A nossa proposta é necessário informar qual o SGDB, sendo que a API identifica a conexão correspondente e utiliza o nome da base de dados, o usuário e a senha.

Quando processam-se os ECLs a API identifica os elementos tratando-os de maneira que componham o dicionário. Isto é realizado da seguinte maneira:

Através do caminho do arquivo do ECL ou do banco de dados ele identifica quais são os elementos existentes e armazena estes separadamente em uma estrutura de registros de apoio (record set), criando hierarquia conforme formato descrito na seção 5.4.4.

5.4.6 Implementação do dicionário na API

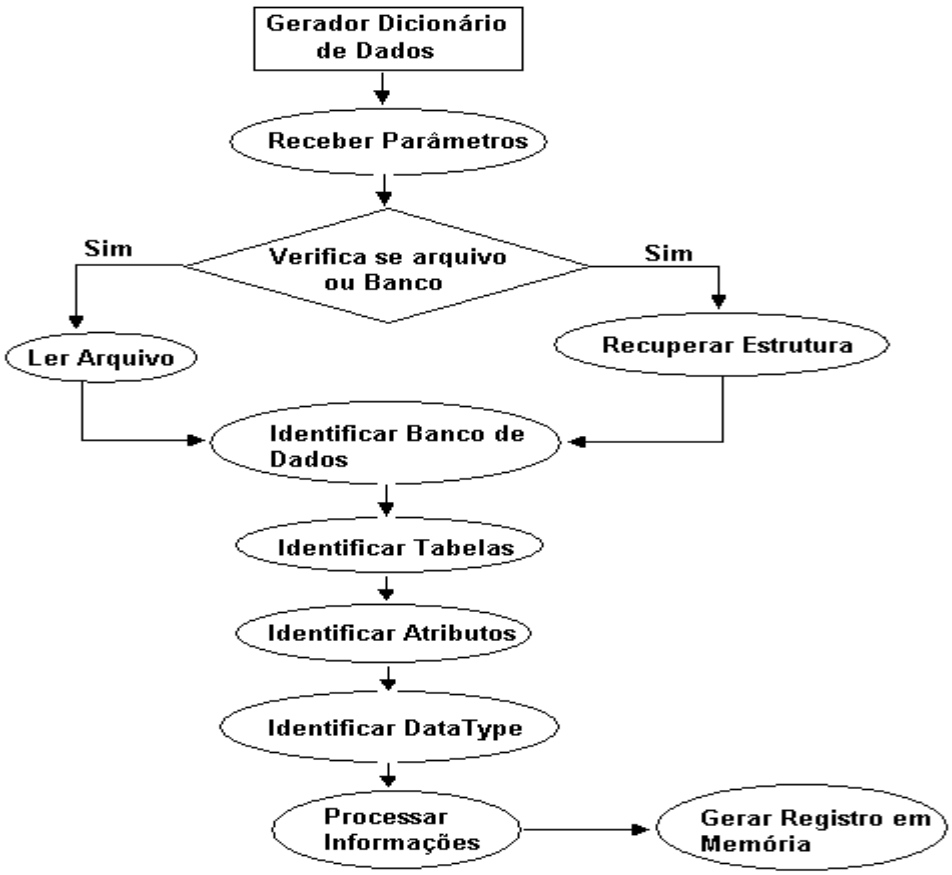


Figura 5.9 - Fluxo do Dicionário de Dados na API

Nesta seção apresentamos o fluxo da geração do dicionário na API na figura 5.9, que é o módulo responsável pela inclusão dos ECLs no dicionário, descrito da seguinte forma:

1) Recebe parâmetros de entrada

A API recebe os parâmetros mostrados da tela da figura 5.8, que podem ser de duas modalidades, um arquivo de dicionário (ECLs) ou a definição da própria base.

2) Testa arquivo → lê arquivo de entrada

A API abre, lê e interpreta o arquivo, segundo as semânticas dos SGDBs, identificando os elementos que compõem o ECL. O arquivo é separado em memória em quatro partes: a primeira é uma coleção das entidades existentes nos ECLs, a segunda é uma coleção dos atributos e suas entidades respectivamente, a terceira é a coleção dos datatypes referentes aos seus respectivos atributos e a quarta uma coleção dos seus relacionamentos. É criada uma estrutura hierárquica para facilitar a manipulação do dicionário, descrita na seção 5.4.4.

3) Testa Base de dados → executa conexão e recupera estrutura da base

A API cria comunicação com a base de dados informada (conexão via driver nativo) e executa a função de reconhecimento de entidades existentes naquela base. Um exemplo de um SGDB *sqlserver* com Stored Procedure de sistema *sp_help,* que verifica a estrutura da base de dados com todas as regras inclusive as constrainsts (integridades referenciais e regras de chaves) existentes.

A API executa a Criação das estruturas de registros de apoio onde ficarão armazenados os elementos identificados na forma hierárquica descrita na etapa 2.

4) Identifica nome do banco

Através da primeira estrutura de registro de apoio o primeiro registro a ser inserido é definido como a base de dados. Na identificação da base de dados é

criada uma estrutura para registrar a *string de conexão* da base específica. Esta string representa o acesso a base de dados solicitada. Ela possui os três principais campos para o acesso que são: o nome da base (Datasource), o nome do usuário (USER), e a senha (PASSWORD) além de qual SGDB referencia tais bases, exemplo Oracle, SQL Server ou outro SGDB componente. Caso tenha sido passado por arquivo de ECLs a API solicitará a fonte de origem (datasource), usuário e senha no momento da identificação da base de dados. Se for autenticado pelo banco de dados não será necessária esta solicitação, pois o mesmo já foi informado no momento do recebimento de parâmetros.

5) Identifica nome da tabela

É através da primeira estrutura de registros de apoio que se identifica às tabelas ou entidades. A estrutura inicial é padronizada para facilitar a manipulação do dicionário.

6) Identifica nome do atributo

A segunda estrutura de registro de apoio identifica os atributos existentes nos ECLs. Estes atributos são separados de forma a reconhecer as tabelas a que pertencem e identificados como filhos destas tabelas.

7) Identifica datatype

Com a terceira estrutura identifica-se os tipos de cada atributo(*datatype*). A terceira estrutura trabalha com os atributos considerando-os como um único conjunto de dados não distinguindo os pais de cada atributo. Em seguida separa-se em uma estrutura os atributos e seus datatypes, exemplo *atributo1.integer*. Caso este atributo seja uma chave ele fica identificado da seguinte forma, exemplo *atributo2.integer.key*.

8) Processa informações e gera a estrutura final do dicionário

O último processo para a geração da estrututra do dicionário é a Verificação de Integridade dos ECLs locais. A API realiza a verificação da integridade relacional dos ECLs locais apresentados. É realizada a verificação dos relacionamentos entres as entidades e se as chaves primárias e estrangeiras foram criadas corretamente na estrutura.

A API executa Criação de Estrutura de Estrutura de Registros de Apoio dos Elementos e Relacionamentos unindo todas as estruturas de registros de apoio criadas anteriormente e gerando uma única estrutura. Nesta estrutura encontra-se todas as informações que o dicionário dispõem e fornece para a integração das bases.

5.4.7 Fluxo do XML Schema do Dicionário

Nesta seção descreve-se o fluxo de geração o XML Schema do Dicionário representado na figura 5.10. É necessário que o Dicionário de Dados esteja processado para que a API gere o XML Schema do dicionário descrito a seguir:

1) Lê o registro processado pelo gerador do dicionário

A partir da estrutura de registros criada na Geração do dicionário gera-se a estrutura do XML Schema do dicionário. Este XML Schema servirá de suporte ao XML Schema de Integração, das bases de dados.

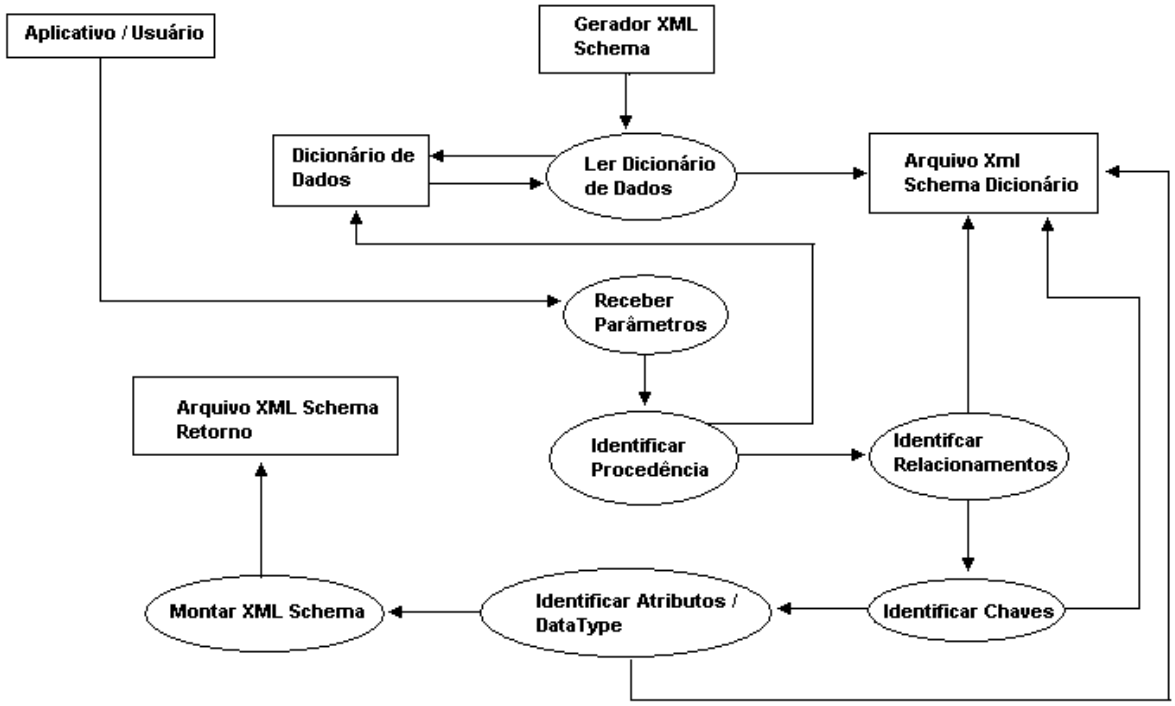


Figura 5.10 - Fluxo da Geração do XML Schema do Dicionário

Seguindo os conceitos apresentados no capítulo 4 sobre o XML Schema gera-se o arquivo XML Schema referente ao dicionário de dados. Este arquivo contém a arvore hierárquica do dicionário de dados com todas as chaves, referências, tipos complexos, elementos e tipos de dados que foram apresentados pelo dicionário.

2) Recebe parâmetros requisitados

Foi criado um Front End para o usuário fornecer os parâmetros necessários a integração, representada na figura 5.11. A API recebe os parâmetros informados sobres SGBD, base de dados, tabelas, atributos e uma condição específica se forem necessário. A API processa estas informações transformando-as em linguagem SQL, através de requisições(queries) independentes para cada base de dados distinta. O resultado deste processamento é armazenado em um primeiro registro de apoio para ser referenciado no retorno dos dados das bases distintas. Após a criação das requisições independentes é aplicado um algoritmo de comparação para a verificação de quais atributos serão relacionados para a integração. As informações dos relacionamentos entre os atributos das bases distintas são

armazenadas em um segundo registro de apoio para sua utilização na geração do XML Schema do dicionário. Com o segundo registro de apoio são criadas as requisições unificadas que farão a recuperação dos dados das bases distintas, e são armazenadas em um terceiro registro de apoio para posterior execução.

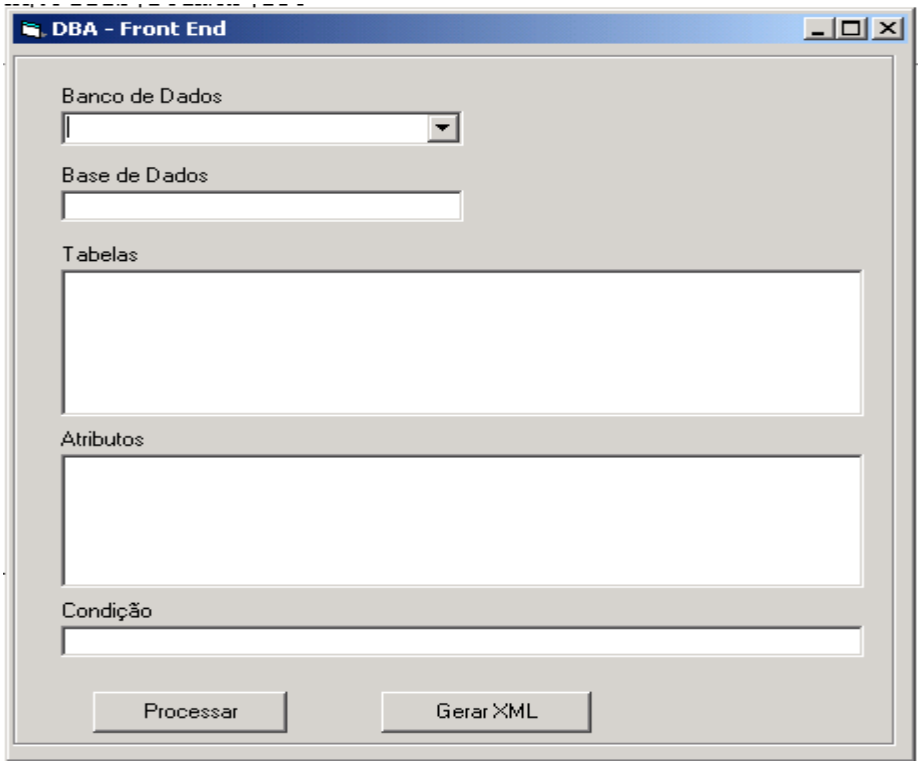


Figura 5.11 - Front End do Usuário para Integração de Bancos de Dados Heterogêneos

3) Identifica no registro de apoio a procedência dos parâmetros

Os parâmetros *tabelas, atributos* e *condição* são confrontados com o dicionário de dados. A API verifica a existência dos parâmetros referenciados pelo usuário dando continuidade no processamento, caso contrário é retornada a mensagem da incompatibilidade dos parâmetros e dicionário.

4) Identifica chave(s) da(s) entidade(s)

<unique name="UnicidadeCodigoPessoa">
<selector>RH/TB_PESSOA/NM_PESSOA</selector>
<field>@CD_PESSOA</field>
</unique>

A identificação das chaves é realizada conforme o parâmetro *tabelas*. As tabelas referenciadas pelo usuário são verificadas pela API no Dicionário retornando quais as chaves devem ser criadas no XML Schema do dicionário consistindo os dados do XML Retorno e não permitindo redundância na criação do arquivo.

A recuperação das chaves cria uma estrutura XML de representação, conforme descrito no capitulo 4 de XML Schema. Nesta representação constrói-se a estrutura do *selector* onde ficam armazenadas as procedências desta chave que é são os bancos de dados e sua tabela respectiva.

5) Identifica relacionamento(s) da(s) entidade(s)

<element name="CD_CODIGO" type="pub:tTB_Endereco" minOccurs="1" maxOccurs="unbounded"/>

<element name="CD_CODIGOTEL" type="pub:tTB_Tlefone" minOccurs="1" maxOccurs="unbounded"/>

Os relacionamentos dos ECLS locais são encontrados no dicionário de dados, mas há a necessidade de representação dos relacionamentos entre entidades de bases distintas sem a necessidade de criação de relacionamentos que unam estas entidades (os relacionamentos das entidades de bases distintas será feito em trabalhos futuros). A API verifica no dicionário, através do parâmetro *condição* quais atributos são equivalentes e referencia-os como chaves virtuais, armazenando em um registro de apoio todas as chaves virtuais para a criação dos *complexTypes* possam ser referenciados corretamente para cada entidade. Este processo permite a API controlar a união das requisições independentes com as requisições unificadas, identificando quais parâmetros serão passados a cada requisição unificada. Verifica-se que um tipo de dado(datatype) do atributo perde sua característica original e passa a pertencer a um tipo complexo(complextype) de maneira que o XML Schema entenda qual é o relacionamento entre o atributo referenciado e o atributo existente no tipo complexo.

6) Identifica atributo(s), datatype(s)

<element name="NM_CELULAR" type="string" />

<element name="NM_NOTEBOOK" type="string" />

A API identifica os atributos referenciados pelo parâmetro *atributos* e verifica no dicionário qual entidade pertence bem como o seu datatype criando a estrutura do complextype no XML Schema.

No registro de apoio dos relacionamentos são incluídos os atributos relacionados, excluindo os demais atributos não havendo redundância de dados.

É verificada a cardinalidade dos atributos através das referências entre as tabelas que estão no dicionário de dados, ou seja poderá existir várias ocorrências no arquivo XML Externo. Esta operação é efetuada por um algoritmo que verifica a cardinalidade entre as entidades dos ECLs e atribui por meio de comparação de atributos similares entre as bases a sua cardinalidade no XML Schema de parâmetros.

7) Cria estrutura XML Schema em meio físico

A API une todos os registros de apoio seguindo a hierarquia do XML Schema e cria o arquivo XML Schema do dicionário de acordo com os parâmetros referenciados pelo usuário.

5.4.8 XML Schema de parâmetros.

O XML Schema de parâmetros é o modelo de estrutura do Arquivo XML Externo onde os dados são recuperados das bases distintas.

As regras aplicadas são as mesmas descritas na seção de XML Schema do dicionário, porém implementadas tendo em vista a integração das bases distintas.

A criação do XML Schema de parâmetros necessita obrigatoriamente dos parâmetros referenciados pelo usuário para a seleção das estruturas dos dados, isto é, precisa-se de um agente externo que informe a estrutura dos parâmetros necessários.

A identificação destes parâmetros no dicionário de ECLs é necessária para a utilização do XML SCHEMA do dicionário recuperar as premissas correspondentes a estes parâmetros.

No XML Schema de parâmetros possui todas as regras de manipulação dos dados encontrados no arquivo XML Externo. Fornece o suporte necessário onde a criação do arquivo XML de retorno que reflete o processo de integração.

5.4.9 XML Retorno ou XMLde Integração

Esta é a última etapa desta arquitetura. Pode-se observar através da figura 5.3 que o Gerador do XML Retorno é a recuperação dos dados dos SGDBs componentes. Isto é possível através do dicionário de ECLs e do XML Schema de parâmetros..

Nesta seção descreve-se o funcionamento de criação do XML Retorno ou de integração como mostra a figura 5.12 e descrito a seguir:

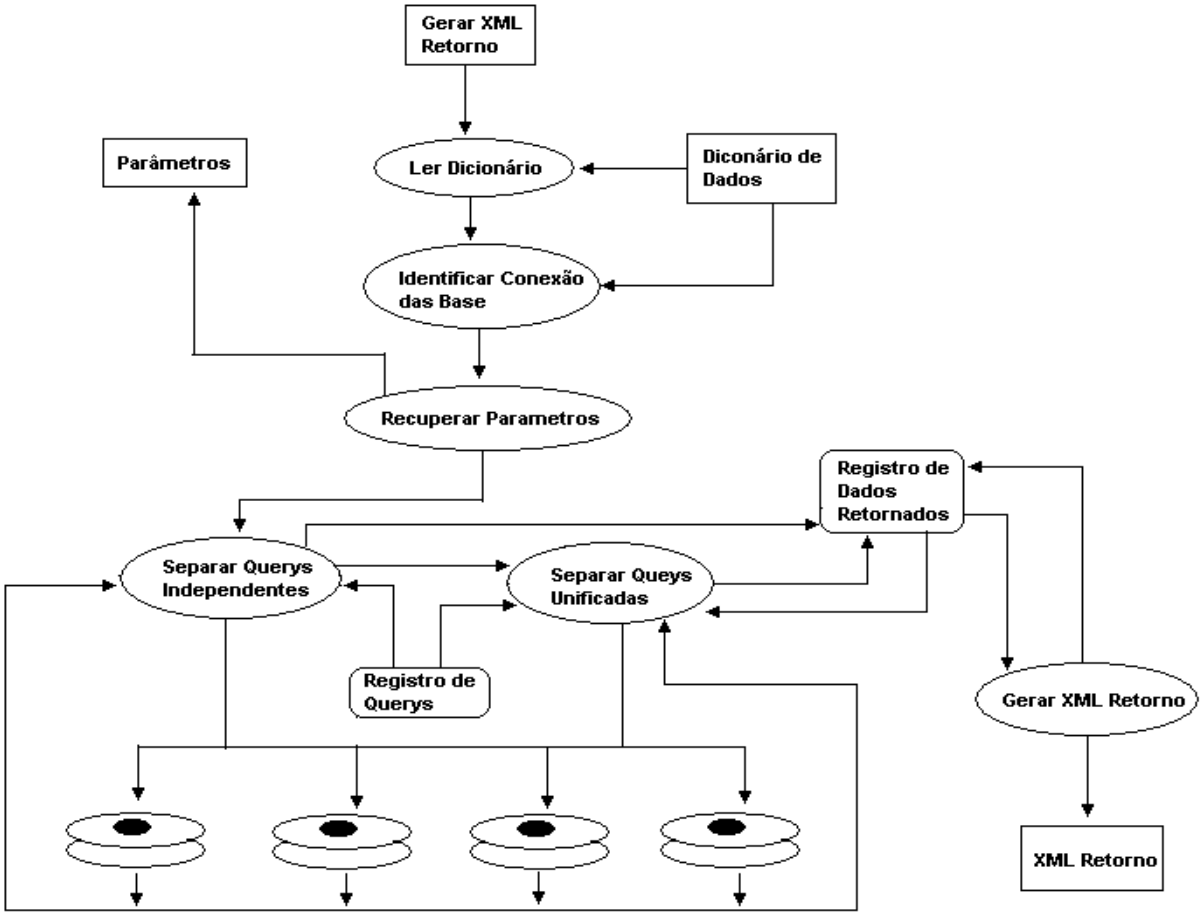


Figura 5.12 - Fluxo da Geração do XML Retorno (XML de Integração)

1) lê o registro processado → Banco(s), Requisições(s), Relacionamento(s).

Os *strings* de conexão são conhecidos, as requisições independentes, unificadas e os relacionamentos são conhecidos, precisa-se recuperar os dados nas bases de dados distintas referenciadas nos parâmetros do usuário e processadas pela API. A API faz a leitura dos registros de apoio correspondentes preparando separadamente cada um para sua execução específica.

2) separa requisições independentes

As requisições independentes são separadas para a execução, de acordo com a base de dados a ser manipulada. A API verifica quais são as bases de dados, e recupera do dicionário o *string de conexão* a ser utilizado, e efetua a conexão. A conexão é feita através de ADODB via driver nativo. A API executa a requisição independente e retorna os dados em uma estrutura de registro de apoio.

3) separa requisições unificadas

As requisições unificadas são separadas para a execução, de acordo com a base de dados a ser manipulada. As requisições unificadas são recuperadas do registro de apoio descrito no passo anterior. As requisições unificadas dependem dos dados retornados das requisições independentes. A API identifica quais são os dados a unificar conforme a condição passada por parâmetro, verifica o registro de dados retornado e cria as requisições unificadas com os respectivos valores.

As requisições unificadas são executadas e armazenadas na mesma estrutura de registros de apoio do retorno das requisições independentes. Este procedimento facilita a criação do XML de retorno, pois como a API retorna apenas um arquivo unificado.

4) criação do arquivo XML Retorno

Para a criação do arquivo XML Externo foi utilizada a camada *XMLPERSIST* que permite a geração automática do XML através das estruturas de registros de

apoio. Esta camada *XMLPERSIST* consiste em informar qual é o tipo de arquivo que deverá ser gerado a partir da estrutura de registro de apoio, sendo esta somente informativa para a Linguagem utilizada.

A Opção por esta camada foi feita por não haver necessidade de criar manualmente a estrutura XML tendo em vista que a mesma segue as regras da W3C e demonstradas no fluxo da API com as descrições das estruturas de registros de apoio.

O Arquivo XML Retorno será salvo em um diretório genérico juntamente com o seu XML Schema respectivo para maior facilidade de acesso.

5.4.10 Benefícios do uso do dicionário de dados manipulados por uma API

Os benefícios decorrem automaticamente da criação do Dicionário de ECLS, embora muito planejamento e esforço tenham que ser despendidos até que se consiga implantar com sucesso um dicionário.

Além do apoio essencial à integração e administração de banco de dados, a API proporciona um local centralizado para acessar todas as informações necessárias referentes à fase de análise e desenvolvimento de sistemas, conforme figura 5.4.

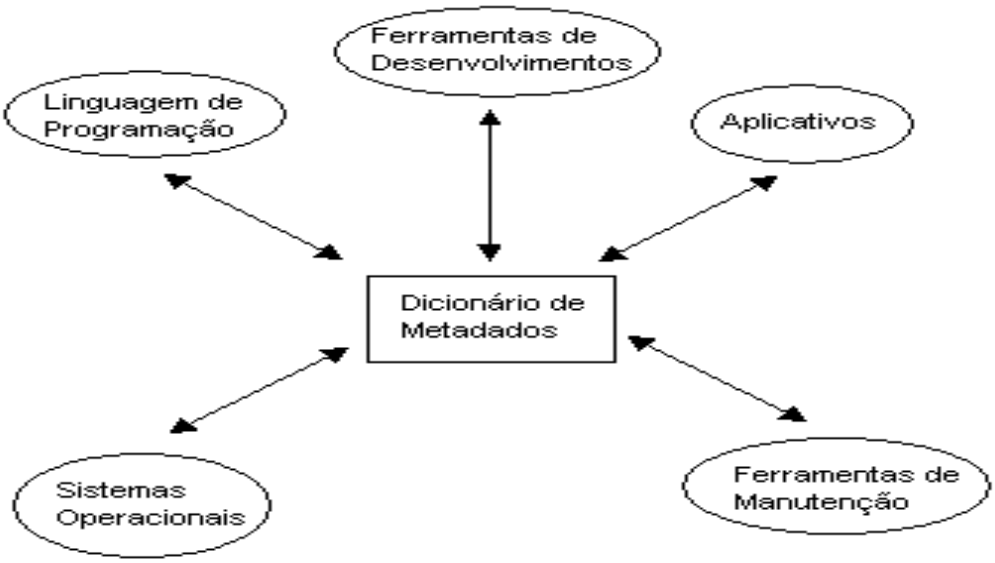


Figura 5.13 - Posicionamento do Dicionário de Dados numa API

Um Dicionário de dados automatizado pela API permitirá responder com rapidez e segurança as seguintes perguntas:

1) Entidades/relacionamentos pertencentes a um determinado cliente/usuário
2) Quais as entidades do cliente que são vistas por um determinado projeto;
3) Quais os atributos da entidade "x" que são vistos pelo projeto "y";

Os objetos manipulados pelo dicionário de ECLs estão intimamente associados ao modelo lógico e físico do banco de dados em particular. Assim, a API tem funções de suportar os processo de programação e projeto físico do banco de dados, tais como geração e manutenção de estruturas de dados e categorias que processam estes dados.

Os requisitos abaixo não estão diretamente associados à metodologia, mas são considerações de nível geral, tornam a API uma ferramenta mais útil e eficiente:

1) Permitir o acoplamento de funções específicas do usuário às funções padrões do dicionário-extensões;
2) On-line e interativo;
3) Interface com diversos SGBDs;
4) Facilidade de uso;
5) Suprir validações e consistências: nomes duplicados, definições incorretas, relacionamentos incompletos;
6) Permitir definição de esquema de segurança de acesso aos metadados;
7) Funções especiais para o administrador do dicionário de dados separadas das funções para usuários normais;

O desenvolvimento da API referencia as regras abordadas neste trabalho, aplicadas a integração de bases de dados heterogêneas. Para melhor entendimento do funcionamento da API descreve-se um estudo de caso discutido no capitulo 6 onde é apresentado um caso prático e aplicada a proposta de integração com a utilização da API.

5.5 TESTES DE ESCALABILIDADE

A escalabilidade nos testes efetuados refere-se ao conceito onde podemos saber se uma aplicação funciona satisfatoriamente em cenários diferentes. Tem-se que ressaltar diferenciando escalabilidade de velocidade, que se refere ao tempo que o servidor demora para responder a uma requisição [CDK00].

Foram realizados testes de escalabilidade para se assegurar de que o método se comporta eficientemente para diversos tipos de *queries* que consultem uma ou várias fontes de dados.

O teste levou em conta o uso de CPU, I/O, Memória, em cada servidor envolvido e atributos e linhas retornadas de integração. Foram utilizadas as ferramentas disponíveis no sistema operacional (Linux Debian) na máquina 4 e nas demais máquinas Windows e SGBD (Oracle e SqlServer) para se medir os recursos.

Uma vez que algumas *queries* utilizam até cinco fontes de dados, foi estimada uma média dos servidores consultados e essa média é a que está presente na tabela 5.1 abaixo.

Requisição/ máquinas envolvidas	**CPU**	**I/O**	**Memória**	**Atributos**	**Linhas Retornadas**	**Tempo de Resposta**
query1/3 máquinas	12%	80%	28%	40	2.300.000	187 s
Query2/4 máquinas	96%	8%	12%	34	400.000	48s
query3/5 máquinas	75%	40%	26%	21	850.000	65s
query4/5 máquinas	10%	72%	43%	11	2.000.000	150s
query5/3 máquinas	30%	61%	30%	18	1.000.000	89s

Tabela 5.1 - Resultados do Teste de Escalabilidade

Como pode-se observar pelos resultados dos testes, o método apresentou-se consistente e de boa escalabilidade. Somente nas *queries* mais complexas houve um alto custo computacional (query2 e query3 por exemplo). Os recursos de I/O

também, de maneira geral, apresentaram-se dentro de valores aceitáveis. Somente a query1 exigiu mais demanda por entrada e saída pois retornou grande quantidade

de dados. Quanto a memória não houve nenhum resultado fora do esperado, que houvesse exceção e interrompesse o processo por demanda de recursos de máquina e encontram-se no padrão de aplicações de Bancos de Dados. As quantidades de atributos das requisições foram listadas para ilustrar a carga de máquina a ser exigida. O tempo de resposta apresentou resultado satisfatório, pois nestes testes não houve a preocupação com a performance das requisições, ou seja não houve customização das consultas.

6 ESTUDO DE CASO E IMPLEMENTAÇÃO

O estudo de caso compreende uma empresa de Assessoria e Consultoria em sistemas chamada GW Informática Ltda, localizada em Curitiba e que efetua serviços para diversas empresas públicas e privadas no Estado do Paraná.

Um dos clientes da GW Informática tinha a necessidade de integrar bases heterogêneas cujo propósito era solucionar problemas internos com divergências de informações nos processos que envolviam banco de dados de Recursos Humanos e banco de dados de Veículos que eram gerenciados por sistemas de informação distintos.

A situação dos processos envolvia funcionários e terceiros contratados que utilizavam veículos da frota para deslocamento a fim de realizarem serviços externos, o que compreendia numa diversidade de despesas como abastecimento, manutenção e outras despesas implícitas como diárias de hotéis, refeições e adicionais. O gerenciamento das informações era executado pelos sistemas específicos de cada área, um para registrar as despesas pessoais dos funcionários e terceiros e outro para registrar despesas com veículos.

A dificuldade de consolidar as informações integrando os bancos de dados distintos com o objetivo de controle efetivo de despesas tanto de pessoas como dos veículos consistia num processo moroso e manual, mesmo sendo auxiliado pelos profissionais da informática da empresa.

Foi proposta uma solução de integração de bases heterogêneas com auxílio de uma API com o propósito de tornar dinâmica a visualização das bases envolvidas e de se tornar uma interface amigável e de fácil manipulação.

O primeiro passo foi o levantamento dos modelos de bancos de dados envolvidos com o levantamento dos metadados.

Segundo Passo foi a criação do dicionário contendo os metadados descrito na seção 5.4.8 na geração do dicionário na API e com layout na figura 5.9, informa-se a ele o banco de dados *RH* que foi modelado pra um Banco de Dados *SQLServer* (Microsoft) e também o banco de dados *VEICULOS*, que foi modelado para um Banco de Dados *ORACLE* e gera-se o dicionário. Na geração do dicionário é

aplicado o mapeamento dos esquemas locais em XML SCHEMAS conforme descrito na seção 5.4.8 para o controle da integridade referencial dos esquemas locais permitindo a API controlar tais referencias de modo a não permitir redundância de dados ou operações inaplicáveis aos bancos de dados, juntamente ao mapeamento é aplicado o mapeamento dos XML SCHEMAS descrito na seção 5.4.9 onde o XML Retorno é gerado. A ferramenta de desenvolvimento usada foi Visual Basic 6 da Microsoft que era a ferramenta disponível na empresa e utilizada por solicitação de seus administradores, pois a manutenção ficaria a cargo da mesma.

Esta decisão foi tomada visto que os administradores não tinham o domínio em ferramentas que manipulam Java, pois a API poderia ser utilizada em qualquer sistema Operacional.

O terceiro passo foi desenvolver um Front End para simular um aplicativo que instanciasse a API, demonstrada na figura 6.1.

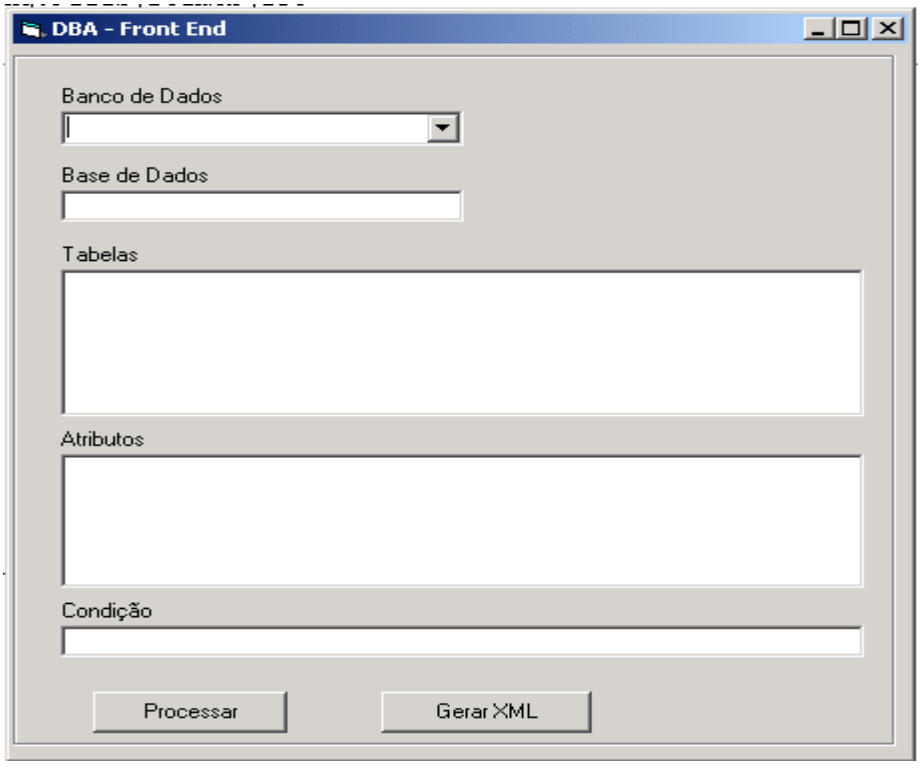


Figura 6.1 - Front End do Usuário para Integração de Bancos de Dados Heterogêneos

O estudo de caso é demonstrado a partir do modelo apresentado na figura 5.5 e 5.6. Estes modelos demonstram a modelagem simplificada de um Banco de Dados de Recursos Humanos e de um Banco de Dados de Veículos respectivamente.

Quarto passo recuperar a placa do veiculo que foi utilizada por determinado funcionário ou terceiro e que tenha permissão de utilizar o veiculo da empresa.

Informamos ao nosso Front End o SGDB SQLServer, informamos a Base de Dados que é o local físico do servidor e informamos: as tabelas *tb_pessoa* e *tb_complemento_pessoa;* atributos: *tb_pessoa.nm_pessoa*; *b_complemento_pessoa.nm_veiculo* e a condição *tb_pessoa.cd_pessoa* = 1. Ao processar, o Front End instancia a API e os ECLs são recuperados do dicionário de dados descrito na seção 5.4.8. Agora informaremos o segundo Banco de Dados envolvido na integração,o Oracle, a tabela *tb_veiculos*, o atributo *tb_veiculos.ds_placa* e a condição *tb_complemento_pessoa.nm_veiculo* = *tb_veiculos.ds_descricao* e processamos as informações. Novamente o Front End instancia a API e carrega os ECLs requisitados do dicionário de dados descrito na seção 5.4.8.

Após concluídos estes dois passos requisitamos ao Front End através do botão Gerar XML apresentado na figura 6.1.

Entretanto antes de gerar o arquivo XML Retorno a API processa a criação do arquivo XML Schema. O Primeiro XML Schema é construído através do Dicionário de Dados onde apresenta todas as regras de XML Schema para a coleção completa do dicionário tratando as referências, chaves e relacionamentos de todos os elementos encontrados no dicionário.

O arquivo do XML Schema do dicionário ficará conforme apresentado na figura 6.2 na página seguinte:


```
- <schema xmlns="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
    <element name="RH_VEICULOS" type="pub:tRHVEI" />
  - <complexType name="tRHVEI">
    - <unique name="UnicidadeCodigoPessoa">
        <selector>RH/TB_PESSOA/NM_PESSOA</selector>
        <field>@CD_PESSOA</field>
      </unique>
    - <unique name="UnicidadeCodigoVeiculo">
        <selector>VEICULOS/TB_VEICULOS/DS_DESCRICAO</selector>
        <field>@CD_VEICULO</field>
      </unique>
    - <sequence>
        <element name="TB_PESSOA" type="pub:tPessoa" minOccurs="1" maxOccurs="unbounded" />
      </sequence>
    - <sequence>
        <element name="TB_VEICULOS" type="pub:tVeiculos" minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Pessoa">
    - <sequence>
        <element name="CD_PESSOA" type="integer" />
        <element name="CD_CODIGO" type="pub:tTB_Endereco" minOccurs="1" maxOccurs="unbounded" />
        <element name="CD_CODIGOTEL" type="pub:tTB_Tlefone" minOccurs="1" maxOccurs="unbounded" />
        <element name="NM_PESSOA" type="string" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Endereco">
    - <complexContent>
      - <extension base="pub:tTB_Pessoa">
        - <sequence>
            <element name="CD_CODIGO" type="integer" />
            <element name="NM_DESCRICAO" type="string" />
            <element name="DT_CRIACAO" type="date" />
            <element name="DS_COMPLEMENTO" type="string" />
          </sequence>
      </extension>
    </complexContent>
  </complexType>
- <complexType name="tTB_TELEFONE">
  - <complexContent>
    - <extension base="pub:tTB_Pessoa">
      - <sequence>
          <element name="CD_CODIGOTEL" type="integer" />
          <element name="DS_DDD" type="integer" />
          <element name="DS_FONE" type="integer" />
        </sequence>
      </extension>
      </complexContent>
    </complexType>
  - <complexType name="tTB_Complemento_Pessoa">
    - <sequence>
        <element name="CD_PESSOA" type="pub:tTB_Pessoa" minOccurs="1" maxOccurs="1" />
        <element name="NM_CELULAR" type="string" />
        <element name="NM_NOTEBOOK" type="string" />
        <element name="NM_BIP" type="string" />
        <element name="NM_VEICULO" type="string" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Veiculos">
    - <sequence>
        <element name="CD_VEICULO" type="pub:tTB_Veiculos" minOccurs="1" maxOccurs="unbounded" />
        <element name="CD_MODELO" type="pub:tTB_Modelo" minOccurs="1" maxOccurs="1" />
        <element name="DS_DESCRICAO" type="string" />
        <element name="DS_PLACA" type="string" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Modelo">
    - <sequence>
        <element name="CD_MODELO" type="pub:tTB_Modelo" minOccurs="1" maxOccurs="unbounded" />
        <element name="DS_MODELO" type="string" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Modelo">
    - <sequence>
        <element name="CD_MARCA" type="pub:tTB_Modelo" minOccurs="1" maxOccurs="unbounded" />
        <element name="DS_MARCA" type="string" />
      </sequence>
    </complexType>
  - <restriction base="string">
      <pattern value="\d{5}-\d{3}" />
    </restriction>
  </schema>
```

Figura 6.2 - O Arquivo XML Schema do Dicionário

Após a geração do Arquivo XML Schema Dicionário de Dados a API executa a criação do XML Independente através dos parâmetros passados conforme descrito acima.

A Figura 6.3 demonstra como ficará o arquivo XML de parâmetros.

```
- <schema xmlns="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
    <element name="RH" type="pub:tRH" />
  - <complexType name="tRH">
    - <unique name="UnicidadeCodigoPessoa">
        <selector>RH/TB_PESSOA/NM_PESSOA</selector>
        <field>@CD_PESSOA</field>
      </unique>
    - <unique name="UnicidadeCodigoVeiculo">
        <selector>VEICULOS/TB_VEICULOS/DS_DESCRICAO</selector>
        <field>@CD_VEICULO</field>
      </unique>
    - <sequence>
        <element name="TB_PESSOA" type="pub:tPessoa" minOccurs="1" maxOccurs="unbounded" />
      </sequence>
    - <sequence>
        <element name="TB_COMPLEMENTO_PESSOA" type="pub:tComplementoPessoa" minOccurs="1" maxOccurs="1" />
      </sequence>
    - <sequence>
        <element name="TB_VEICULOS" type="pub:tVeiculos" minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Pessoa">
    - <sequence>
        <element name="NM_PESSOA" type="string" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Complemento_Pessoa">
    - <sequence>
        <element name="NM_VEICULO" type="pub:tTB_Veiculos" minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  - <complexType name="tTB_Veiculos">
    - <sequence>
        <element name="DS_DESCRICAO" type="string" />
        <element name="DS_PLACA" type="string" />
      </sequence>
    </complexType>
  - <restriction base="string">
      <pattern value="\d{5}-\d{3}" />
    </restriction>
  </schema>
```

Figura 6.3 - XML De Parâmetros

Como mostra a figura 6.4 pode-se observar a união das bases pelo complexType tTB_Complemento_Pessoa no seu element *nm_pessoa* que se tornou um tipo de tTB_Veículos onde sua referencia é o element *ds_descricao*.

Isto foi possível pela ultima condição passada pelo Front End, onde foram informados os atributos de filtro para e recuperação dos dados *nome da pessoa*, *nome do veiculo* e *placa do veiculo*.

A API então vai agora efetivamente recuperar os dados das bases selecionadas e retornar o resultado obtido em um arquivo XML demonstrado na figura 6.4.

```
- <xml xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882" xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
    xmlns:rs="urn:schemas-microsoft-com:rowset" xmlns:z="#RowsetSchema">
  - <s:Schema id="RowsetSchema">
    - <s:ElementType name="row" content="eltOnly" rs:updatable="true">
      - <s:AttributeType name="NM_PESSOA" rs:number="1" rs:write="true">
          <s:datatype dt:type="string" rs:maybenull="false" />
        </s:AttributeType>
      - <s:AttributeType name="NM_VEICULO" rs:number="2" rs:write="true">
          <s:datatype dt:type="string" rs:maybenull="false" />
        </s:AttributeType>
      - <s:AttributeType name="DS_PLACA" rs:number="2" rs:write="true">
          <s:datatype dt:type="string" rs:maybenull="false" />
        </s:AttributeType>
        <s:extends type="rs:rowbase" />
      </s:ElementType>
    </s:Schema>
  - <rs:data>
    - <rs:insert>
        <z:row NM_PESSOA="JOAO ANTONIO FIGUEROA" />
        <z:row NM_VEICULO="CHEVROLET-CORSA-009" />
        <z:row DS_PLACA="AAT6666" />
      </rs:insert>
    </rs:data>
  </xml>
```

Figura 6.4 - XML Retorno (Integração)

Este arquivo XML Retorno poderá ser utilizado agora por qualquer aplicação que suporte ambiente XML Esta tecnologia auxiliará muitas empresas que contenham operações em banco de dados distribuídos, facilitando assim a interoperabilidade dos dados, com o apoio da API para que todas as suas bases sejam reconhecidas e possam ser integradas. Os dados em um ambiente padrão XML permite a manipulação do usuário dos dados retornados, tomando assim

decisões cada vez mais eficazes perante a administração da informação existente no ambiente da empresa.

7 CONCLUSÃO E TRABALHOS FUTUROS

A integração de banco de dados é o aperfeiçoamento da técnica que administra recursos de informação descentralizados.

No contexto de sistemas multidatabases heterogêneos, o administrador depende primordialmente de uma definição sobre o acesso permissível as diversas fontes de dados, operações para consolidar os esquemas, concordar nas relações semânticas entre esquemas de banco de dados e satisfazer integridades referenciais e inerentes.

A integração proposta neste trabalho contempla a coordenação de fontes distintas de banco de dados e sugere uma solução que permite que a integração de multidatabases heterogêneos com a criação de um dicionário de metadados em uma API, onde decisões de resolução de conflito sejam feitas baseadas em objetivos localizados, integridades referenciais, estratégia de integração e conhecimento de correspondência de interesquema que precisam e não sejam conhecidos globalmente.

A primeira contribuição deste trabalho é o desenvolvimento de uma ferramenta de integração que apresenta ao administrador de banco de dados ou do ambiente de sistemas aplicativos as descrições dos bancos de dados a serem integrados. Um dicionário de dados automatizado pela API permitirá responder com rapidez e segurança quais entidades/relacionamentos pertencentes a um determinado cliente/usuário, quais entidades do cliente que são vistas por um determinado projeto, quais os atributos das entidades e a quais bancos de dados pertencem são vistos pelos projetos.

Os objetos manipulados pelo dicionário de dados estão intimamente associados ao modelo lógico e físico do banco de dados em particular. Assim, a API tem funções de suportar os processo de programação e projeto físico do banco de dados, tais como geração e manutenção de estruturas de dados e categorias que processam estes dados.

A segunda contribuição é a representação do resultado através das facilidades da metalinguagem XML, onde cada componente de banco integrado ou a

integrar é tratado de maneira distinta e dinâmica onde o administrador de banco de dados ou do ambiente de sistemas aplicativos especifica todas as equivalências entre esquemas a serem integrados. O XML Schema Retorno criado neste trabalho pode ser considerado uma extensão do modelo XML Schema da Microsoft, pois, tem o adicional da identificação dos bancos de dados.

A terceira contribuição para as equipes de desenvolvimento de aplicações é a visualização dos bancos de dados e atributos baseados na API, que facilitará o trabalho nos projetos de aplicações e possíveis integrações.

Apresentou-se um estudo de caso que envolve toda a dinâmica de integração iniciando pelo levantamento dos metadados dos modelos, criação da API, utilização da mesma para mapear os esquemas locais com o tratamento devido das semânticas envolvidas e por último a apresentação em XML de integração, onde os sistemas envolvidos farão o tratamento dos dados que são fundamentais para a seqüência de execução de seus processos.

As tecnologias recentes de tratamento de visões de dados e consultas (como XML) podem fazer a diferença no re-direcionamento de recentes pesquisas relacionadas à área, como a extensão de metodologias de integração, unificação de processos e interfaces para a Web e o desenvolvimento de sistemas inteligentes para a integração de esquemas.

O conhecimento e a informação se tornaram um bem precioso para a organização, sendo assim, é necessário gerenciá-lo bem para se ter um efetivo controle sobre este patrimônio. Este gerenciamento só é possível através de uma arquitetura coesa com o pleno domínio das bases de dados. O seu valor aumenta considerando que as informações estão cada vez mais integradas e passam a dar apoio a tomadas de decisão a fim de obter vantagens competitivas para a organização.

Trabalhos futuros a serem desenvolvidos na área de integração de banco de dados: 1) criação de uma ferramenta case; 2) tratamento da semântica dos multidatabases heterogêneos que sugestionem os DBAs ou usuários especializados nas integrações; 3) criação de controle de atualizações, onde as alterações efetuadas nos bancos de dados sejam refletidas no dicionário de ECLs. 4) Criação de alias para identificar as bases dos sistemas gerenciadores de banco de dados, tornando transparente o acesso.

REFERÊNCIAS BIBLIOGRÁFICAS

[Ab96] ABITEBOUL, S. **Foundations of Banco de dados**. California: Addison-Wesley Publishing Company, 1996.

[ACM97] Abiteboul, S.; CLUET; MILO, T.. **"Correspondence and Translation for Heterogeneous Data"**, Proceedings of the International Conference on Database Theory: ICDT-97, Delphi, Greece, p. 352-363, 1997.

[ABS00] S. Abiteboul, P. Buneman e D. Suciu. **Data on the Web: from Relations to Semistructured Data and XML**. 1ª ed, Morgan Kaufmann Publishers, San Francisco, California, USA, 2000.

[Ah91] AHMED, R. *et al.* The PEGASUS heterogeneous multidatabase system. **Computer**, p. 19-27, dec. 1991.

[BGL+99] C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakostatinou, P. Velikhov e V. Chu. **XML-Based Information Mediation with MIX**. In Proc. of ACM SIGMOD Conf. on Management of Data, p. 597-599, Philadelphia, Pennsylvania, USA, Junho 1999.

[BL84] BATINI, C.; LENZERINI, M. A metrhodologi for data Schema integratrion in the entity-reletionship model. **IEEE Trans. Software Eng.**, v. SE-10, n. 6, nov. 1984.

[Bel88] BELCASTRO, V. *et al.* An overview of the distributed query system DQS. *In*: **Lecture Notes in Computer Science**, SpringerVerlag, v. 303, p. 170-189, 1988.

[BMT98] BIJNAGTE, L.; MUSSO, D.; TOBACK, B. **Real Estate Transaction Standard Protocol Specification**. USA, 1998. Disponível em: http://www.rets-wg.org/docs/ retsproto.pdf. Acessado em: 06/2001.

[Bos01] BOSAK, J. **XML, Java, and the future of the Web**. USA, 1997,. Disponível em: http://www.ibiblio.org/pub/sun-info/standards/ XML/why /XMLapps.htm. Acessado em: 11/2001.

[Car94] CARDENAS, A. Heterogeneous distributed database management: The HDDBMS. **Proceedings of the IEEE**, 75(5), p. 588-600, may 1987.

[CR94] CARNEIRO, S.; RAMESH, V. Schema Design Conceitual colaborador: Um Modelo de Processo e Sistema de Protótipo. **Tech. informe**, departamento de MIS, Univ. de Arizona, 1994.

[CDK00] COULOURIS, George; DOLLIMORE, Jean and KINDBERG, Tim. "Distributed Systems: Concepts and Design". 3 ed., Addison-Wesley, 2000.

[Cha01] CHANG, B. **Oracle XML**. Rio de Janeiro: Campus, 2001.

[CDF+02] CHANNEL DEFINITION FORMAT. USA, 1997. Disponível em: http://www.w3.org/TR/ NOTE-CDFsubmit.html. Acessado em: 03/2002.
[CM+02] CHECKFREE & MICROSOFT. **Open Financial Exchange**. USA, 1997. Disponível em: http://www.ofx.net/ofx/default.asp. Acessado em: 02/2002.

[CCL91] CHU, W.W.; CHEN, Q.; LEE, R. Uma Aproximação Padrão-baseada por Derivar Aproximado e Intensional Answers. **Procedimentos em Interoperability em Sistemas de Multidatabase**, p. 262-265, 1991.

[Chu90] CHUNG, C. DATAPLEX: An access to heterogeneous distributed databases. **Communications of ACM**, 33(1), p. 70-80, jan. 1990.

[Co98] COHEN, W.W. Integração de Bancos de dados Heterogêneos Withoud Domínios Comuns que Usam Questões baseado em Semelhança de Testual. **SIGMOD Record**, v. 27, n. 2, p. 201-212, jun. 1998.

[CMM91] COLMILHO, D.; MARTELO, J.; MCLEOD, D. A Identificação e Resolução de Heterogeneidade Semântica em Sistemas de Multidatabase, IMS'91 Procedimentos. **Primeiro Seminário Internacional em Interoperability em Sistemas de Multidatabase**, p. 136-43, 1991.

[Col97] COLOMB, R.M. Impacto de Heterogeneidade Semântica em Federar Bancos de dados. **O Diário de Computador**, v. 40, n. 5, p. 235-244, 1997.

[Com+02] CONSORTIUM, ELECTRONIC BUSINESS CARD. USA, 1997. Disponível em: http://www.imc.org/pdi/vcardwhite.html. Acessado em: 02/2002.

[Co02] COVER, R. **The XML Cover Pages - Bioinformatic Sequence Markup Language BSML**. USA, 2001. Disponível em: http://www.oasis-open.org/cover/bsml.html. Acessado em: 02/2002.

[Da01] DATE, C.J. **Introdução a Sistemas de Banco de Dados**. Tradução da 7. ed. Americana. Rio de Janeiro: Campus, 2001.

[DH84] DAYAL, U.; HWANG, H. View definition and generalization for banco de dados integration in multibase: A system for heterogeneous distributed banco de dadoss. **IEEE Trans. Software Eng.**, v. SE-10, n. 6., nov. 1984.

[De83] DEMO, B. Program analysis for convertion from a navigational to a specification database interface. *In*: **Proceedings of the 9th Int. Conference on Very Large DataBases**, Florence, p. 387--398, oct. 1983.

[DR88] DURNFORD, A.; RUSSELL, J. **Rights in Technical Data and Computer Software**. Oct. 1988.

[DL87] DWYER, P.; LARSON, J. Some experiences with a distributed database testbed system. **Proceedings of the IEEE**, 75(5), p. 633-647, may 1987.

[Elm90] ELMAGARMID, A.K.; PU, C. A Introdução de Editores de Convidado para o Assunto Especial em Bancos de dados Heterogêneos. **ACM Computing Pesquisas**, v. 22, n. 3, p. 175-181, set. 1990.

[EHW] ELMASRI, R.; HEVNER, A.; WELDREYER, J. The category concept: An extencion to the entity-reletionship model. **Data and Knowledge Eng. J.**, v. 1, n. 1, p. 75-116, june. 1985.

[EFD01] EXTENSIBLE FORMS DESCRIPTION LANGUAGE. USA, 1998. Disponível em: http://www.w3.org/TR/NOTE-XFDL. Acessado em: 12/2001.

[Fur97] FURLAN, J.D. **Modelagem de Negócios**. São Paulo: Mckron Books, 1997.

[Gan96] GANESH, M. Regras de Entidade-identificação mineiras para Integração de Banco de dados. Procedimentos de KDD-96. **Segunda Conferência Internacional em Descoberta de Conhecimento e Dados que Minam**, p. 291-4, 1996.

[GRE98] Green, Dale. **The Java Development Kit** (JDK) 1.2 Reviewer´s Guide, Java Business Expo in New York, 1998

[Gui02] GUILLAW, D. **Astronomical Markup Language**. USA, s.d. Disponível em: http://monet.astro.uiuc.edu/~dguillau/these. Acessado em: 04/2002.

[Hal+02] HALL, R. **Oracle Corporation**. Disponível em: www.oracle.com. Acessado em: 08/2000.

[Ham+02] HAMILTON, J.M. **Developing a Business Case for an XML Authoring System**. USA, IDEAlliance Dec 2001. Disponível em: http://www.idealliance.org/papers/ XML2001papers/tm/WEB/03-01-05/03-01-05.htm. Acessado em: 05/2002.

[HM78] HAMMER, M.; MELEOD, D. The semantic data mode: A modeling mechanism for banco de dados applications. *In*: **Proc. ACM SIGMOD Conf.**, p. 26-36, 1978.

[HC90] HAYNE, S.; CARNEIRO, S. Multiusuário Visão Integração Sistema (MUVIS): Um Sistema Especialista para Integração de Visão. **Procedimentos da Sexta Conferência Internacional em Dados Criar**, p. 402-410, 1990.

[HM85] HEIMBIGNER, D.; MCLEOD, D. A federated architecture for information management. **ACM Transactions on O#ce Information Systems**, 3(3), p. 253-278, jul. 1985.

[HF95] HERRIN, E.H.; FINKEL, R.A. **Schema and Tuple Trees:** An Intuitive Structure for Representing Relational. 1995.

[Hsi92] HSIAO, D. Tutorial on federated databases and systems (Part I). Metodologias para Conversão de Esquemas em SBDHs 53. **The VLDB Journal**, 1(1), p. 127-179, jul. 1992.
[Hul87] HULL, R.; KING, R. Semantic database modeling: Survey, applications, and research issues. **ACM Computing Surveys**, 19(3), p. 45-62, sep. 1987.

[HRM+02] HUMAN RESOURCE MANAGEMENT MARKUP LANGUAGE. Disponível em: http://www.XML.org/XML/zapthink/std116.html. 2000. Acessado em: 05/2002.

[HUT89] HUTCHINSON, S.A. *et al*. Planning sensing strategies in a robot work cell with multisensor capabilities. **IEEE Transactions on Robotics and Automation**, v. 5, n. 6, p. 765-83, dec. 1989.

[GGRS00] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shin, "XTRACT: A System for Extracting Document Type Descriptors from XML Documents", SIGMOD 2000, pages 165-176, Dallas, Texas, May, 2000.

[GSN99] G. Gardarin, F. Sha e T. Ngoc. **XML-based Components for Federating Multiple**
Heterogeneous Data Sources. In Proc. of Int'l Conf. on Conceptual Modeling, p. 506-519, Paris,
France, Novembro 1999.

[La83] LARSON, J. Bridging the gap between network and relational management systems. **Computer**, p. 82-92, sep. 1983.

[LE02] LENZERINI, M.. **Data Integration**: A Theoretical Perspective. In PODS, pages 233–246, 2002.

[LNE86]_____; NAVATHE, S.B.; ELMASRI, R. A theory of attribute equivalence in banco de dadoss with application to Schema integration. (ex tender version) Honeywell Systems Develoment Division Minneapolis. **MN. Tech. Rep.**, CSC. 86-10: 8212, s.d.

[LD97] LI, S.; DANZIG, P.B. Boolean Semelhança Medidas para Descoberta de Recurso. **Transações de IEEE em Conhecimento e Dados Criar**, v. 9, n. 6, p. 863-876, nov./dec. 1997.

[Li97] LI, W.S. Integração Semântica em Bancos de dados Heterogêneos que Usam Redes de Neural. **Procedendo da 20ª Conferência de VLDB**, 1994.

[LK01] LIBERTY, J.; KRALEY, M. **Desenvolver Documentos XML para WEB**. Tradução americana por Flávia Cruz. São Paulo: Makron Books, 2001.

[Li81] LIEN, Y. Hierarquical schemata for relational databases. **ACM Transactions on Database Systems**, 4(1), p. 107-131, 1981.

[Lim90] LIMA, J.V. de. An integration of structuded documents into dbms. In: **Proceedings of Eletronic Publishing'90**, Gaithersburg,md - Usa: 1990.

[Lis75] LISP. Function Tracing Package (Trace), M.I.T., Project MAC, Cambridge, Mass., 1975.

[LA86] LITWIN, W.; ABDELLATIF, A. Multidatabase interoperability. **Computer**, 19(12), p. 10-18, dec. 1986.

[LOS03] Lóscio, BF, Managing the Evolution of XML-based Mediation Queries, tese de doutorado, Universidade Federal de Pernambuco, 2003.

[MAZ96] MADHAVARAM, M.; ALI, D. L.; ZHOU, M. Integrando sistema de banco de dados distribuído heterogêneo. **Computadores & Engenharia Industrial**, v. 31, n. 1-2, p. 315-318, oct. 1996.

[Mal99] MALHOTRA, A. **XML Schema Requirements**. São Paulo: Makron Books, 1999.

[OV99] M. Ozsu e P. Valduriez. **Principles of Distributed Database Systems**. Prentice Hall, 2ª ed,
1999.

[OV01] OZSU, M.T. **Princípios de Sistemas de Banco de Dados Distribuídos**. Tradução da 2. ed. Americana por Vanderberg D. de Souza. Rio de Janeiro: Campus, 2001.

[PM88] PECKHAM, J.; MARYANSKI, R. Semantic data models. **ACM Computing Surveys**, 20(3), p. 153-189, sep. 1988.

[Pfa99] PFAFFENBERGER, B. **Web Publishing with XML**. Virginia: Academic Press, 1999.

[RB97] RAMESH, V.; BATE, S. Integração de constrangimento de Integridade em Bancos de dados Heterogêneos: Uma Metodologia Aumentada para Integração de Schema. **Sistemas de Informação**, v. 22, n. 8, p. 423-446. 1997.

[RS+01] RANDELL, B.; SKONNARD, A.A. **Guide to XML and Its Technologies, Microsoft**. USA, sep 1999. Disponível em: http://msdn.microsoft.com/library/default.asp?url=/
library/en-us/dnXML/html/XMLguide.asp. Acessado em: 12/2001.

[RPRG94]REDDY, M.P.; PRASAD, B.E.; REDDY, P.G.; GUPTA, U.M. Uma metodologia para integração de bancos de dados heterogêneos. **Transações de IEEE em Conhecimento e Dados que Criam**, v. 6, n. 6, p. 920-33. (Dec. 1994).

[Red02] REDMOND, W. **Microsoft XML Architect to Unveil XML in "Office 11" At XML Conference & Exposition 2002**. Disponível em: www.microsoft.com. Acessado em: 25/10/2002.

[Re+01] REFSNES, J.E. **Introduction to DTD - An introduction to the XML Document Type Definition**. USA, 1999. Disponível em: http://www.XML101.com/dtd/dtd_intro.asp. Acessado em: 12/2001.

[SFL94] SACHS, J.; FERRAIOLO, K.; LANDOLL, D. Advocacy for an Engineering Process Oriented Approach to Assurance. **Position paper submitted to the Invitational Workshop on IT Assurance and Trustworthiness**, jan. 1994.

[SEP+94] SEVERANCE; ENBODY; PURDY. **Evolving Dynamic Load Balancing to Shared Memory Parallel Processors, to appear as a poster session in Supercomputing '94**. Disponível em: http://www.egr.msu.edu/~crs/papers/load_94/.

[SL90] SHETH, A.P.; LARSON, J. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. **ACM Computing Surveys**, 22(3), p. 183-236, sep. 1990.

[Sh] SHIPMAN, D.W. The functionl data model and de data language daplex. **ACM Trans Banco de dados Sym.**, v. 6, n. 1, p. 140-173, s.d.

[SKS99] SILBERSCHATZ, A.; KORTH, H.F.; SUDARSHAN, S. **Sistemas de Banco de Dados**. São Paulo: Makron Books, 1999.

[Sil01] SILVA, O.J. **XML Aplicações Práticas**. Rio de Janeiro: Érica, 2001.

[SOP02] SIMPLE OBJECT ACCESS PROTOCOL. USA, 2000. Disponível em: http://www.w3. org/TR/SOAP/. Acessado em: 03/2002.

[Sim02] SIMPSON, J.E. **XML and Web Sites**. Disponível em: www.XML.com. Acessado em 10/03/2002.

[SZ87] SISTLA, A.P.; ZUCK, L.D. On the eventuality operator in temporal logic. *In*: **Proceedings, Symposium on Logic in Computer Science**, Ithaca, New York, p. 153-166, jun. 1987.

[Soa92] SOARES, J. Approximating Euclidean Distances by Small Degree Graphs. **Technical Report 9205**, University of Chicago, 1992.

[Sp94] SPACCAPIETRA, S.; PAI, Christine. Integração de Visão: um passo adiante resolvendo conflitos estruturais. **Transações de IEEE em Conhecimento e Dados que Criam**, v. 6, n. 2, p. 258-74, abr. 1994.

[SD91] STEPHEN, J.H.; DAVID, J.B. What connectionist models learn. *In*: RICHARD, J.M.; ZEEVI, Y. **Neural networks:** theory and applications. Harcourt Brace Jovanovich, 1991.

[Sy02] SYNCHRONIZED MULTIMEDIA INTEGRATION LANGUAGE. USA, 2000. Disponível em: http://www.w3.org/AudioVideo/. Acessado em: 06/2002.

[TM87] TEMPLETON, M. *et al.* MERMAID: a frontend to distributed he terogeneous databases. **Proceedings of the IEEE**, 75(5), p. 695-708, may 1987.

[TC00] TSENG, F.S.C.; CHIANG, J.J.; Yang, W.P. Integração de Relações com Estruturas de Schema Contraditórias, Sistema de Banco de dados Heterogêneo. **Dados e Conhecimento Criando**, v. 27, p. 231-248. s.d.

[Urb91] URBANO, S.D. Um Vigamento Semântico para Ambientes de Banco de dados Heterogêneos, IMS' 91 Procedimentos. **Primeiro Seminário Internacional em Interoperability em Sistemas de Multidatabase**, p. 156-63. 1991.

[VBO01] V. Vidal, A. Brayner e A. Oliveira. **Self-Maintenance of Materialized Views in XMLBased**
Mediator. In Informal Proc. of 1th Int'l Workshop on Data Integration Over the Web
(DIWeb), p. 32-46, Interlaken, Switzerland, Junho 2001.

[VLS01] V. Vidal, B. Lóscio e A. Salgado. **Using Correspondence Assertion for Specifying the**
Semantics of XML-Based Mediators. In Proc. of Workshop on Integration of Information on the
Web, Rio de Janeiro, Abril 2001.

[Wal99] WALSH, N. **Understanding XML Schemas**. Disponível em: www.w3c.com. Acessado em: 01/07/1999.

[Yon00] YOUNG, M. **XML Step by Step**. USA: Microsoft Press, 2000.

[Yu91] YU, C. *et al*. Determinando Relações entre Nomes em Bancos de dados Heterogêneos. **SIGMOD Record**, v. 20, n. 4, p. 79-80. Dec. 1991.

[Za79] ZANIOLO, C. Design of Relational Views over Network Schemas. **SIGMOD Conference**, p. 179-190, 1979.

[Zh97] ZHAO, J.L. Coordenação de Schema em administração de banco de dados federada: uma comparação com integração de Schema. **Sistemas de Apoio de Decisão**, v. 20, n.3, p. 243-57, jul. 1997.