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

Liluyoud Cury de Lacerda

UM MODELO DE MAPEAMENTO DE ESTRUTURAS E DADOS RELACIONAIS PARA DOCUMENTOS XML

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

Prof. Dr. Murilo Silva de Camargo

Florianópolis, Fevereiro de 2001.

UM MODELO DE MAPEAMENTO DE ESTRUTURAS E DADOS RELACIONAIS PARA DOCUMENTOS XML

Liluyoud Cury de Lacerda

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

_____

Fernando Álvaro Ostuni Gauthier, Dr.

Banca Examinadora

_____

Murilo Silva de Camargo, Dr. (Orientador)

_____

Roberto Willrich, Dr.

_____

Fábio Paraguaçu Duarte da Costa, Dr.

_____

Darlene Figueredo Borges Coelho, Drª.

A Deus e a minha família.

À minha família, por todo incentivo, apoio e carinho. Em especial aos meus pais, Djalma Xavier de Lacerda e Rosa Libaneza Cury de Lacerda, pelo amor e paciência que demonstraram ao longo de toda a minha vida.

Enfim, agradecer a todos que, direta e indiretamente, participaram do momento especial de realização deste trabalho.

SUMÁRIO

FIGURAS

RESUMO

As trocas de informações entre os vários tipos de sistemas de banco de dados se dão atualmente de diversas maneiras, onde até então não se havia um padrão. Usam-se muito *drivers* específicos de comunicação e protocolos proprietários e diversos outros meios com o objetivo de integrar e trocar informações entre diferentes sistemas de bancos de dados. Com a crescente necessidade de compartilhamento de informações entre sistemas, pessoas e organizações, a tendência é usar protocolos abertos e altamente compatíveis com a internet para intercâmbio dessas informações (principalmente para comércio eletrônico), e por esse motivo pode-se dizer que o XML é o mais indicado, dado sua capacidade de processar (estruturar, organizar, armazenar e recuperar) conjuntos complexos de informações, como arquivos multimídia, estruturas hierárquicas de dados e imagens digitais, entre diversos outros tipos de documentos, assim como dados em estruturas relacionais ou baseadas em objetos, tudo isso com velocidade, confiabilidade e escalabilidade sem igual. Dado essas premissas, essa dissertação propõe um modelo de conversão de estruturas de dados relacionais, que é o modelo mais comum atualmente, para documentos XML e vice-versa.

ABSTRACT

The changes of information among the several types of database systems are made in several ways, where there was not a standard way. It is used a lot of specific drivers of communication and protocols proprietors and etc. with the objective of to integrate and to change information among different systems of databases. With the coming of the internet, and the globalization of the economy (mostly for e-commece), the tendency is to use open protocols and highly compatible with the internet for information exchange, and for that reason it can be said that XML is the most suitable, given your capacity to process (to structure, to organize, to store and to recover) groups compounds of information, as files multimedia, hierarchical structures of data and digital images, among other several types of documents, as well as data in structures relate or based on objects, all this with speed, reliability and unique scalability. Given those premises, that dissertation proposes a model of conversion of structures of relational data, that it is now the most common model, for documents XML and vice-versa.

Capítulo 1

INTRODUÇÃO

Observa-se uma evolução considerável nos sistemas de armazenamento de dados nas últimas décadas. Trocou-se o acesso seqüencial de arquivos pelos sistemas gerenciadores de banco de dados (SGBD). Esses sistemas trouxeram grandes benefícios em relação aos antigos meios de armazenamento e acesso a dados. Porém, com o avanço da tecnologia e o aumento da complexidade e heterogeneidade das aplicações, novas necessidades surgiram e começaram a ficar evidentes as limitações que esses sistemas gerenciadores de bando de dados traziam. Duas destas limitações são:

- *Integração com outros SGBDs:* A maioria deles tem soluções proprietárias de acesso, gerenciamento, armazenamento e manipulação de dados. Essa característica praticamente impossibilita a integração dessas bases de dados com os de outros fabricantes de forma nativa e automática. O que é feito por alguns SGBDs, é o fornecimento de alguns *adapters/links* de comunicação para outros SGBDs, mas esses *adapters/links* são muito ineficientes no que se refere a velocidade, compatibilidade e principalmente na utilização plena dos recursos oferecidos pelo outro sistema gerenciador de banco de dados.

- *Falta de um padrão para comunicação entre SGBDs:* Mesmo havendo uma tentativa incessante de se definir um padrão de comunicação (ODBC, OLEDB, COM, DCOM, CORBA), cada fabricante tenta impor sua tecnologia. Algumas soluções de *Datawarehouse*[1] quebram essas dificuldades de integração e comunicação, mas são muito complexas de se implantar e a um custo muito alto, o que impossibilitaria a sua aplicação em organizações de pequeno e médio porte.

Estas limitações dificultam muito a construção de certas aplicações que envolvem uma grande integração entre diferentes tipos de bancos de dados, baseado em

[1] Datawarehouse é como se fosse um grande armazém de dados que tem como objetivo armazenar dados de diferentes bases a fim de que possam ser analisados.

um grande número de restrições de integridade ou que definem políticas específicas de uma empresa ou organização.

Nos anos 70 começaram a surgir os primeiros projetos de linguagens e sistemas que tinham como objetivo incluir nos SGBDs tradicionais mecanismos para realização de processos automáticos sem a interferência do usuário. Mas estes projetos só foram reconhecidos e começaram a tomar corpo nos anos 90 com o advento de soluções de datawarehouse [CD91]. Os SGBDs que são acrescidos desta capacidade, de automaticamente realizar tarefas (comunicar) com outros SGBDs, estão hoje em plena utilização e constante aperfeiçoamento, mas como foi dito antes, ainda é uma solução de difícil implementação e implantação, e relativamente de custo elevado.

Nas seções seguintes serão introduzidos os conceitos necessários para o desenvolvimento desta dissertação. Não é objetivo aqui descrever todos os pontos que envolvem a integração de SGBDs, nem descrever sistemas específicos, mas apenas dar material suficiente para a discussão que será apresentada, que seria modelagem de estruturas relacionais em documentos XML e vice-versa.

1.1 – Porque o Uso da XML como Padrão de Integração?

1.1.1 – Apresentação

Com as constantes mudanças dos ambientes organizacionais e empresariais, as operações de negócios ficaram mais descentralizadas organizacional e geograficamente, já que a competição tem crescido a nível global, onde a demanda dos usuários e do mercado favorecem o estilo de gerenciamento descentralizado. Microcomputadores têm aumentado seu poder de processamento, o que encorajou o crescimento de redes locais com dados e informações locais.

Em contra partida, pela própria filosofia de um SGDB (compartilhar dados), ficou difícil a integração das informações espalhadas ao longo das diversas redes locais que fazem parte de uma organização ou modelo de negócio.

Antes, com banco de dados centralizados, não havia problemas com integração, mas tinham problemas com a degradação de performance[1], alto custo com a manutenção de sistemas de banco de dados de mainframe[2] e problemas de confiança criados pela dependência de um local central (atualmente não se recomenda que dados sejam guardados em um único local, pois afetaria a segurança dessas informações, entre diversos outros motivos) .

Ao distribuir as informações ao longo de diversos bancos de dados, pode-se agilizar o acesso aos mesmos, já que eles tenderão a estar mais perto do usuário (dados devem estar localizados perto de locais de maior demanda), haverá um processamento de dados mais rápido dado a divisão da carga de trabalho, facilitará futuras expansões já que se pode adicionar novos locais sem afetar outros já existentes, reduzirão os custos de funcionamento em relação a mainframes, interface do usuário mais amigável, menos perigo de uma queda de um ponto de dados, e principalmente, acarretará uma independência de plataforma[3], já que diferentes plataformas podem acessar dados entre si em locais diferentes.

Entretanto, essa independência de plataforma aumenta a complexidade de gerenciamento e controle, diminui a segurança em determinados níveis, e a falta de padrões compromete a integração dos dados, e é justamente este último tópico que esse trabalho irá discutir com detalhes.

1.1.2 – Propósito e Importância

O principal objetivo é desenvolver e apresentar um modelo de mapeamento entre estruturas relacionais para um padrão altamente aberto e preparado para a internet, nesse caso a XML, o que possibilita a interação de bases heterogêneas e autônomas, o que é essencial para se desenvolver aplicações de alta abrangência e complexidade.

Recentemente, as necessidades dos usuários são as mais variadas e as necessidades das aplicações envolvendo banco de dados estão ficando cada vez mais amplas e complexas, o que aumenta a demanda de acesso à fonte de dados heterogênea.

---

[1] Locais cada vez mais distantes acessando remotamente os dados centralizados acarreta problemas sérios de performance.
[2] Isso é discutível se levar em consideração o Custo Total de Propriedade (TCO) dos equipamentos de pequeno e médio porte.
[3] Refere-se ao conjunto de equipamentos, softwares e sistemas que rodam em um determinado local

Assim, a integração das informações de sistemas autônomos em ambiente distribuídos e heterogêneos se tornou assunto de grande importância atualmente.

Até agora, o estudo na integração de banco de dados distribuídos e heterogêneos tem apontado para cada componente tecnológico, como solucionar conflito de *schema*, e processamento de consultas distribuídas, e há poucos casos de estudo que adaptam esta tecnologia para a aplicação. Além disso, não foi construída nenhuma metodologia padrão (sistemática) para adaptar essas aplicações aos diversos bancos de dados existentes, muito menos criado um modelo de referência único [HK99].

Atualmente existem vários métodos de desenvolvimento e modelos de referência para criar aplicações (produtos adaptáveis) usando métodos de integração de banco de dados heterogêneos e distribuídos, e conseqüentemente existem diversos estudos sobre o desenvolvimento de sistemas de informações integrados baseados em banco de dados heterogêneos. Como resultado desses estudos é visto que:

- Há uma enorme tendência em investigar tecnologias que envolvem sistemas de múltiplos bancos de dados, baseando-se principalmente na investigação dos métodos de transformação dos dados, gateways entre banco de dados e aplicações, abordagem utilizada na interface de comunicação, analisando a mediação dos mesmos e apontando seus méritos e deméritos.

- Foram desenvolvidos vários modelos de referência de sistemas de informação integrado baseado em banco de dados distribuídos e heterogêneos, analisando sua performance, modelo de mediação e referência a múltiplas bases de dados.

- Foram desenvolvidos vários métodos de conexão a múltiplos bancos de dados usando método de interface de objeto (sendo os mais comuns o CORBA e DCOM). No desenvolvimento de métodos híbridos (usando várias abordagens), devem-se analisar os prós e contras de cada método de conexão, fazendo assim uma análise mais consistente[1].

[1] Como resultado dessa dissertação foi desenvolvido um sistema baseado no modelo híbrido, no caso CORBA (Commom Object Request Broker Architeture) e DCOM (Distributed Commom Object Model).

- Foram produzidas muitas tecnologias de integração de banco de dados, usando modelos como o DOM (ajuda na integração e na representação de dados de produtos heterogêneos) e SAX (é a interface padrão para o DOM), que são mais conhecidos1, ou qualquer outro modelo.

- Foram produzidos vários ambientes de desenvolvimento de aplicações em ambiente de múltiplos bancos de dados, cada um com suas particularidades.

Serão abordados conceitos básicos na qual essa dissertação se baseia para demonstrar meios de poder integrar múltiplos bancos de dados via XML. Detalhes desses conceitos e tecnologias serão discutidos nos capítulos posteriores.

1.1.3 – XML está Preparado?

Se o HTML é uma linguagem de descrição de páginas, pense na XML como uma linguagem universal de descrição de dados. Em XML é possível descrever numa metalinguagem, o conteúdo e o formato que determinados arquivos de dados terá. Em última análise, a XML permite descrever a semântica dos dados. Quais as vantagens? *Independência de plataforma tecnológica e interoperabilidade plena*[2].

Os cenários de aplicação são inúmeros. Pense numa arquitetura de integração de aplicativos de internet a sistemas legados de empresas, na qual as transformações semânticas dos dados de mainframe para web sejam realizadas por um servidor de metadados XML.

Ou ainda, pense num repositório público e único na internet, de especificações XML, contendo descrições de documentos que empresas aceitam receber de outras companhias, como pedidos, cancelamento de pedidos e fichas de fornecedor. Agora imagine que sistemas podem coletar esses arquivos XML e interoperar com sistemas de outras empresas, de forma automática. Sem que nenhuma conheça detalhes dos sistemas da outra. Por fim, leve este último exemplo para o contexto de integração de banco de dados (que é o discurso desta dissertação) e B2B[3] e verá que a aderência ao padrão XML, pode ser um fator acelerador na consolidação de mercados onde os legados estejam conectados on-line.

---

[1] A serem discutidos com mais detalhes a seguir, nessa dissertação.
[2] É o que se deseja mostrar nessa dissertação.
[3] Business to Business: Negócios realizados entre empresas via internet

1.2 – Objetivos desta dissertação

Esta dissertação tem como objetivo apresentar um modelo de mapeamento de estruturas e dados relacionais para documentos XML e vice-versa, o que resolveria vários problemas de comunicação e interação entre banco de dados heterogêneos, já que a XML está definida em uma plataforma aberta e amplamente aceita por praticamente todas as empresas do mundo.

Para tanto será definida uma estreita ligação entre o conceito de integração de dados e tecnologia XML. Serão apresentados modelos de mapeamento entre dados e documentos XML e vice-versa, que é essencial para justificar a proposição dessa dissertação.

Será criado um ambiente virtual de simulação de integração de sistemas gerenciadores de banco de dados, com o objetivo de demonstrar os conceitos a serem apresentados.

Tal ambiente será construído baseado na linguagem de programação Delphi 5.0, componentes de comunicação CORBA da Visibroker, tecnologia ADO da Microsoft[1], e os sistemas gerenciadores de banco de dados SQL Server 2000, Access 2000, Interbase 6.0 entre outros repositórios de dados como Paradox, dBase, Excel e arquivos textos.

Vale lembrar que não serão abordados conceitos novos, no que se refere a integração ou modelagem, apenas serão analisados dois conceitos já existentes (SGBDs Relacionais e XML) visando a união das duas tecnologias. Essa abordagem, (o uso simultâneo das duas tecnologias) é que se pode considerar como nova.

1.3 – Organização da Dissertação

Para atingir tais objetivos, este trabalho foi dividido em 6 capítulos.

[1] CORBA (*Commom Object Request Broker Architeture*) e DCOM (*Distributed Commom Object Model*) foram escolhidos por serem as tecnologias mais presentes no mercado, além de serem gratuitas e bastante funcional. Os bancos dados abordados também são de grande presença, entretanto outros SGBD como Oracle, Sybase, DB2, etc. não foram testados nessa dissertação por não estarem disponíveis para avaliação e estudo.

O primeiro e o último capítulo são a introdução e a conclusão, respectivamente, onde na conclusão são apresentados resultados, pontos fortes e fracos e perspectivas futuras.

No capítulo 2 são apresentados conceitos mais detalhados da XML, abordando basicamente suas especificações, que é o suficiente para entender os conceitos a serem descritos dessa dissertação.

No capítulo 3, delinear-se-á uma relação entre banco de dados relacionais e XML, descrevendo basicamente o que uma tecnologia (XML) pode ajudar na melhora da eficiência de um conceito (integração e comunicação).

Uma vez concluídos esses capítulos, no capítulo 4 será apresentada uma forma de se mapear as informações encontradas em banco de dados relacionais para documentos XML e vice-versa.

No capítulo 5, será apresentado um modelo de middleware desenvolvido para apresentar o modelo de mapeamento de dados relacionais para documentos XML descrito no capítulo 4.

Como complemento a essa dissertação, foi incluído dois apêndices, sendo que o primeiro (Apêndice A), fala de softwares middlewares que facilitam a interação entre SGBDs Heterogêneos via XML, e no segundo (Apêndice B), apresenta uma plataforma de intercomunicação de informações via XML que é atualmente considerado estado da arte e finalmente no Apêndice C tem a apresentação de um procedimento formal de mapeamento de documentos XML para banco de dados e vice-versa, baseado no modelo apresentado no capítulo 4.

Capítulo 2

EXTENSIBLE MARKUP LANGUAGE (XML) 1.0

Como já fora explanado anteriormente nessa dissertação, a XML (*eXtensible Markup Language* ou Linguagem de Marcação Estendida) é um subconjunto da SGML (*Standard Generalized Markup Language* ou Linguagem de Marcação Padrão Generalizada) que permite que uma marcação específica seja criada para definir idéias e compartilhá-las na rede e na web. Estendendo essa idéia de modo a atender o objetivo dessa dissertação, também se podem mapear dados e informações a partir de diferentes bancos de dados para documentos XML de uma maneira um tanto simplista, mas eficiente.

Mas antes de definir a maneira de como se dará esse mapeamento, é primordial entender primeiramente um pouco de XML, para justificar a proposição deste trabalho. O objetivo deste capítulo é justamente este, desbravar os conceitos que envolvem a especificação XML[1].

2.1 – O que é XML

2.1.1 - Apresentação

XML é uma linguagem de marcação para documentos contendo informações estruturadas, isto é, mais uma maneira de se representar uma informação.

Apesar de ter a palavra “linguagem” em seu nome, pode-se dizer que XML não é uma linguagem específica, pois não define um vocabulário de comandos e não define uma gramática, define apenas algumas regras mínimas.

Complementando, XML é uma especificação que determina regras para a criação de documentos genéricos que acrescentam informação semântica ao texto através de <marcadores> e são estruturados, facilitando a manipulação, pesquisa e extração de dados.

[1] Este capítulo é baseado nas especificações XML do W3C.

Uma informação estruturada pode ter conteúdos diversos (palavras, imagens, som, vídeo, etc.) e a indicação de como é o comportamento deste conteúdo (por exemplo, uma determinada informação pode ser tratada tanto como um cabeçalho, um rodapé ou uma legenda, dependendo de qual indicação é usado, e ao pensar no contexto dessa dissertação, esse conteúdo pode significar diferentes informações em um banco de dados). Quase todos os documentos que são produzidos atualmente têm alguma estrutura.

Uma linguagem de marcação, como o XML, é um mecanismo muito eficiente de identificação de estruturas em um documento. Atualmente o número de documentos que são baseados em XML, é absurdamente grande (considerando que a XML é uma especificação relativamente nova).

Para o propósito desta dissertação, a palavra "documento" refere-se não somente aos tradicionais documentos, como este trabalho aqui, mas também a uma variedade de outros formatos de dados. Isto inclui gráficos vetoriais, transações de comércio eletrônico, equações matemáticas, dados, metadados[1], servidores API[2] e milhares de outros tipos de informações estruturadas.

2.1.2 Por que XML?

A XML não especifica nenhuma semântica ou conjunto de tags. Na realidade, XML é uma meta-linguagem para descrever linguagens de marcação ou estruturas de dados. Em outras palavras, XML provê facilidades para definir *tags* e a estrutura de relacionamentos entre eles.

Já que não se tem um conjunto pré-definido de *tags*, não pode haver nenhuma semântica pré-concebida. Toda a semântica de um documento XML será tanto definida pelas aplicações que os processam quanto pelas folhas de estilo (*stylesheets*[3]).

O XML é fruto de uma especificação mais antiga, o SGML – *Standard Generalized Markup Language* (Linguagem de Marcação Generalizada Padrão) definido pela ISO 8879 [BM00]. O SGML tem sido o padrão por décadas para manter

[1] Metadado é um dado que provê informações sobre um determinado dado.
[2] Application Program Interface (Interface de Programa de Aplicação): Define protocolos de comunicação para determinada aplicação.
[3] São folhas de estilos que é um meio de determinar como os dados serão apresentados em um determinado documento.

repositórios de informações estruturadas independente de fabricante, mas não é bem apropriado para servir documentos ao longo da Web (por várias razões técnicas que estão fora do escopo desta dissertação).

Definir a XML como um perfil de aplicação do SGML significa que qualquer sistema em conformidade com o SGML, está habilitado a ler documentos XML[1]. Entretanto, usar e entender documentos XML não requer um sistema que seja capaz de entender a generalidade completa do SGML (que provê uma estrutura arbitrária e de difícil implementação para ser usado em um navegador web ou aplicações voltada a internet). Sistemas inteiramente SGML resolvem grandes e complexos problemas, o que justificaria sua aplicação, mas manipular documentos estruturados fornecidos pela web raramente requisitaria tal poder (é muito comum acontecer quando se quer disponibilizar conteúdo SGML na web, criar um filtro desse conteúdo para XML, padronizando assim procedimentos para negócios via internet [HTR00].

É pelos motivos expostos anteriormente, que não é indicado o uso de SGML para o propósito de nossa dissertação.

2.2 – Características da XML

2.2.1 – Objetivos

Segundo as recomendações da própria W3C, o conjunto de especificações XML se baseia nos seguintes objetivos:

- *Deve ser direto o uso do XML na Internet:* Usuários deverão visualizar documentos XML tão facilmente e rapidamente quanto os documentos HTML. Na prática, isso somente será possível quando navegadores XML estiverem tão robustos e largamente difundidos quanto os navegadores HTML, mas o princípio continua.

- *XML deverá suportar uma grande variedade de aplicações:* A XML deve ser benéfica para uma diversidade enorme de aplicações (criação, navegação, análise de conteúdo, manipulação de dados, etc.). Embora o foco inicial seja

[1] Existem sutis diferenças entre documentos entendidos por sistemas XML e sistemas SGML, mas os mesmos são facilmente tratados.

servir documentos estruturados na web, isto não significa que se deve definir estreitamente a XML.

- *XML deve ser compatível com SGML:* A maioria das pessoas envolvidas no "esforço XML" vem de organizações que tiveram um grande envolvimento com assuntos voltados ao SGML. XML foi desenvolvido pragmaticamente para ser compatível com padrões existentes enquanto resolve os problemas envolvendo o envio de documentos estruturados pela web.

- *Deve ser fácil desenvolver programas que processam documentos XML:* O meio mais direto de expressar este objetivo é que foi possível a partir das especificações XML apenas, desenvolver um programa para essa dissertação que processa documentos XML[1].

- *O número de características opcionais na XML deve ser o mínimo possível, senão zero:* Características opcionais inevitavelmente aumentam problemas com compatibilidade, principalmente quando usuários que queiram compartilhar documentos se deparam com algumas confusões e frustrações (problemas de padronização)2.

- *Documentos XML devem ser humanamente legíveis e racionalmente limpos:* Mesmo que não se tenha um navegador XML, ao receber documentos XML, qualquer pessoa deve estar habilitada a olhar o conteúdo desse documento em qualquer editor de textos, e prontamente identificar o significado do conteúdo.

- *O desenvolvimento da XML deve ser preparado rapidamente*: Esforços padronizados são notoriamente vagarosos, entretanto quando houve a "necessidade" da XML, o mesmo foi rapidamente desenvolvido, assim como acontece com suas atualizações.

[1] Será discutido no Capítulo 5, um exemplo de programa que processa documentos XML a partir de tabelas de SGBDs Relacionais.
[2] A Microsoft tenta implementar um modelo de XML diferente do que esta sendo definido pelo W3C, alegando melhora na eficiência da linguagem.

- *O desenvolvimento da XML deve ser formal e conciso:* Essencialmente significa que a XML deve ser expressa em EBNF1 e deve ser amena a ferramentas e técnicas de compilação moderna (a gramática SGML não pode ser definida em EBNF, já que escrever um tradutor SGML requer o manuseio de uma variedade de características de linguagem raramente usadas e de difícil tradução).

- *Documentos XML devem ser facilmente criados:* Embora devam existir editores sofisticados para criar e editar conteúdo XML, é possível criar documentos XML diretamente em um editor de textos com simples comandos e scripts.

- *A concisão em marcadores XML é de mínima importância:* Muitas das características de documentos SGML foram desenvolvidas para minimizar a quantidade de digitação requerida. Essas características não são suportadas na XML, já que suportar essas características adiciona um considerável fardo para os tradutores SGML (o que não seria interessante para a XML).

2.2.2 – Componentes da XML

A seguir serão abordados, de forma bem concisa, alguns dos principais conceitos da especificação XML.

2.2.2.1 – Documentos

Documentos XML são arquivos de texto Unicode[2] contendo marcadores, conteúdos e obedecendo a uma hierarquia de elementos a partir de uma raiz. A seguir é mostrado um exemplo de documento XML que identifica os dados pessoais de um professor.

```
<?xml version="1.0" encoding="utf-8"?>
<professor>
  <nome>Liluyoud Cury de Lacerda</nome>
  <foto href="imagens/liluyoud.jpg" />
  <endereco tipo="Residencial">
    <logradouro>Rua Jamary, 1931</logradouro>
    <bairro>Pedrinhas</bairro>
```

[1] Extended Backus-Naur Form (EBNF) é um conjunto de regras, chamadas produções. Cada regra descreve um fragmento específico de sintaxe. Um documento é válido se o mesmo pode ser reduzido a uma única regra específica. Maiores detalhes ver em *Compilers: Principles, Techniques, and Tools* de Aho, Sethi, e Ullman ou qualquer outro livro sobre compiladores modernos.
[2] Unicode é um padrão que permite aos computadores representar e manipular, de forma consistente, texto de qualquer sistema de escrita existente.

```
    <municipio>Porto Velho</municipio>
    <uf>RO</uf>
  </endereco>
  <telefone tipo="Celular">(69) 3229-8969</telefone>
  <email>liluyoud@hotmail.com</email>
</professor>
```

Figura 1 – Um documento XML simples

Percebe-se que é um documento texto de fácil entendimento e auto-explicativo, onde as informações foram dispostas de forma estruturada e com significado (graças aos marcadores).

O documento começa com uma instrução de processamento: `<?xml ...?>`. Esta é a declaração XML. Embora não seja obrigatória, a sua presença explícita identifica o documento como um documento XML e indica a versão da XML com a qual ele foi escrito.

2.2.2.2 – Elementos

Elemento é a forma de marcação mais comum, delimitados pelos sinais de menor (<) e maior (>). A maioria dos elementos identifica a natureza dos conteúdos que envolvem.

O primeiro elemento de um documento XML é chamado de elemento raiz (ou nó raiz, pois um documento XML pode ser representado como árvore) e pode haver elementos vazios ou não:

```
Elementos vazios
<professor></professor> ou <professor />

Elementos não vazios
<professor>Liluyoud Cury de Lacerda</professor>
```

Figura 2 – Sintaxe de um elemento XML

2.2.2.3 – Atributos

São utilizados para qualificar um elemento. São pares de valores nomeados que ocorrem dentro das marcas de início após o nome do elemento seguindo a estrutura `nomeDoAtributo="valorDoAtributo"`, onde todos os valores de atributos devem estar entre aspas.

```
<professor tipo="Substituto">
```

Figura 3 – Sintaxe de um atributo de um elemento XML

2.2.2.4 – Referências a Entidades

Em XML, alguns caracteres foram reservados para utilização, por exemplo, o sinal de maior “>”. Caso seja necessário usar esses caracteres reservados é necessário fazer uma referência a eles usando uma seqüência de caracteres que deve começar com “&” (e comercial) e terminar com “;” (ponto-e-vírgula). O sinal de maior é representado pela seqüência “`>`”.

Cada entidade deve ter um nome único, “`>`” só pode referenciar o caractere maior e nenhum outro. Essas referências podem usar formas numéricas que podem ser referências decimais, “`&#8478;`”, e referências hexadecimais, “`&#x211E;`”, sendo que ambas se referem ao caractere unicode número `U+211E`.

2.2.2.5 – Comentários

Comentários são textos em um documento XML que não são analisados por um processador XML. São usados apenas para ilustrar algum conteúdo dentro de um documento XML.

Comentários iniciam com `<!--` e terminam com `-->`, podem conter qualquer dado e podem ser colocados em qualquer lugar do documento, desde que não estejam dentro de um marcador.

```
<professor>Liluyoud Cury de Lacerda</professor>
<!-- Esses dados estão desatualizados -->
<endereco>Córrego Grande – Florianópolis - SC</endereco>
```

Figura 4 – Exemplo de comentários em XML

2.2.2.6 – Instruções de Processamento

É uma forma de fornecer informações a uma aplicação e tem a forma `<?nomeDaInstrução dadosDaInstrução?>`.

```
<?xml version="1.0" encoding="utf-8"?>
```

Figura 5 – Exemplo de instruções de processamento

No exemplo anterior têm-se uma instrução que se chama `xml` e os dados que complementam a instrução é a versão do documento XML e o tipo de codificação unicode.

2.2.2.7 – Seções CDATA

Seções CDATA são usadas para inserir em um documento XML qualquer conjunto de caracteres (até mesmo os reservados), e faz com que o analisador os ignore. Uma seção CDATA e delimitado pelo marcador de início "`<![CDATA[`", e pelo de término "`]]>`".

Do início ao fim de uma seção CDATA, todos os caracteres ali existentes são passados diretamente a aplicação, sem passar pelo interpretador XML. Pode ser passado em uma seção CDATA, por exemplo, um conjunto de caracteres hexadecimais que representam uma imagem (ao ser passado a aplicação, a mesma pode tratar essa informação contida no CDATA como uma imagem).

```
<foto><![CDATA[FFFA012FFACD348728FFDAS87191271ADAS1271E]]></foto>
<caracteresReservados><![CDATA[<>=*&GT;]]></ caracteresReservados >
```

Figura 6 – Exemplo de seção CDATA

2.2.2.8 – Declaração de tipos de documentos

Declaração de tipos de documentos (DTD) é usada para passar ao analisador informações sobre o tipo de conteúdo de um documento e de seus elementos.

Apesar da XML permitir certa liberdade na criação de seu conteúdo, a declaração de tipos de documentos pode ser usada para restringir o tipo de informação (conteúdo) que pode ser usado em um determinado documento XML.

```
<!ELEMENT album (fotos*)>
<!ELEMENT fotos EMPTY>

<!ATTLIST album id   ID    #REQUIRED>
<!ATTLIST album nome CDATA #REQUIRED>
<!ATTLIST fotos href CDATA #REQUIRED>
```

Figura 7 – Exemplo de seção DTD

Baseado no DTD acima se pode afirmar que o XML abaixo é válido:

```
<album id="lcl01" nome="Férias SC">
  <foto href="20010001.jpg" />
  <foto href="20010002.jpg" />
</album>
```

Figura 8 – XML válido em relação a um DTD

Uma maneira bem mais eficiente para definir o tipo de conteúdo em um documento XML é o uso de XML *Schema*, que apesar de não ser tão simples quanto o DTD, suporta *namespaces* e permite definição de novos tipos.

O uso de esquemas, seja DTD ou XML *Schema*, é essencial para que haja comunicação usando XML, definindo assim padrões a serem seguidos na troca de informações.

2.2.3 – Documentos XML válidos e bem formados

Existem duas categorias de documentos XML, os bem formados e os válidos.

2.2.3.1 – Documentos bem formados

Um documento somente pode ser definido como sendo bem formatado se ele obedece às especificações da XML e segue as regras mínimas de sintaxe, isto é, documentos que não são bem formados não são documentos XML.

Algumas dessas regras seriam:

- Ter um, e apenas um, elemento raiz
- Valores dos atributos estarem entre aspas ou apóstrofes
- Atributos não se repetirem dentro de um mesmo elemento
- Todos os elementos terem etiqueta de fechamento
- Elementos estarem corretamente aninhados

Por definição, pode-se dizer que não há documento XML que não seja bem formatado.

2.2.3.2 – Documentos válidos

Documentos XML válidos, são documentos que baseados em um DTD ou XML Schema (mais usado atualmente), obedece a todas as suas restrições de declaração (seqüência e aninhamento de elementos, atributos e seus valores, tipos de conteúdo de elementos, etc.).

A validação refere-se à comparação de um documento com um determinado esquema[1] (regras/restrições), e por isso um documento bem formado pode ser válido em uma aplicação e não ser válido em outra, pois a validade é definida em um esquema (DTD ou XML Schema).

Um documento XML é considerado válido em relação a um esquema se obedecer todas as suas regras.

A validação de um documento XML é muito importante para a viabilização da comunicação entre aplicações, pois se criam padrões de documentos. É comum que aplicações requeiram documentos válidos para poder processá-los, pois garantem que a informação ali passada segue todas as regras previamente estabelecidas.

Uma aplicação típica pode esperar que os elementos de um documento XML façam parte de um vocabulário limitado, que tenham certos atributos com valores e tipos definidos, cujos elementos sejam organizados de acordo com uma determinada estrutura hierárquica, etc., e essa é a função de um esquema XML.

2.2.4 – Especificações Relacionadas

Desde sua criação, várias "tecnologias relacionadas" têm surgido para facilitar a manipulação de XML e complementar suas especificações, dentre elas pode-se citar:

2.2.4.1 – DOM

Document Object Model (DOM) é uma recomendação que provê formas de acesso aos dados estruturados utilizando scripts, permitindo aos desenvolvedores interagir e computar tais dados consistentemente.

DOM é uma API (Applications Programming Interface) independente de plataforma e linguagem que é utilizada para manipular as árvores do documento XML (e HTML também).

Esse foi o modelo escolhido para a leitura dos documentos XML na aplicação resultante dessa dissertação, dado sua riqueza de funções e principalmente por permitir a manipulação direta da árvore XML.

[1] Esquema: modelo que descreve todos os elementos, atributos, entidades, suas relações e tipos de dados.

2.2.4.2 – SAX

SAX (Simple API for XML) é baseado em eventos (eventos analisados são reportados para as aplicações através de chamadas callback). SAXs permitem construções "extendidas" da árvore XML. O SAX utiliza bem menos memória e é mais rápido (no que se refere a processamento da árvore XML) que o DOM, mas em contrapartida, não tem tantos recursos (funções) quanto o DOM.

2.2.4.3 – XSL

Extensible Style Language (XSL)[1] define uma linguagem de folhas de estilo padrão para a XML. O XSL apresenta duas seções: a linguagem de transformação (XSLT) e a formatação de objetos (XSL-FO).

A linguagem de transformação pode ser usada para transformar documentos XML em algo agradável para ser visto, assim como transformar para documentos HTML, e pode ser usada independentemente da segunda seção (formatação de objetos).

A apresentação de um documento XML em um determinado formato (Pdf, HTML, etc.) é dependente de uma folha de estilos.

Figura 9 – Folhas de Estilo

2.2.4.4 – XQuery

É uma linguagem declarativa de consulta baseada em operadores e no modelo de dados, similar ao SQL. Pode realizar pesquisas sobre um ou mais documentos XML e pode construir novos documentos a partir dos resultados de uma seleção. É ideal para pesquisa, mas não suporta *updates* ou *inserts*.

A seguir é feito uma relação do *XQuery* com o XSLT, já que ambas são linguagens para extração e transformação de dados.

[1] Maiores especificações em *http://www.w3.org/TR/WD-xsl*

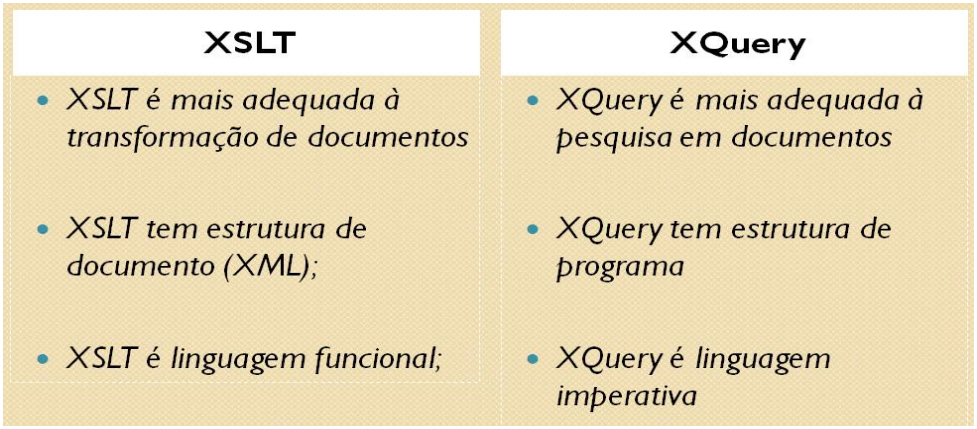


Figura 10 – Diferenças entre XSLT e XQuery

Pode-se dizer que em tese, tudo o que se pode fazer com *XQuery* se pode fazer com XSLT.

2.2.4.5 – XPath

XPath é usada para representar caminhos para os nós de um documentos XML e divide-se em três partes: modelo de dados; linguagem de expressões; e operadores e funções.

Essencialmente, XPath é uma linguagem usada para navegar na árvore XML, onde uma expressão XPath é um caminho na árvore que resulta em um valor (número, texto, booleano), objeto (elemento, atributo, nó de texto) ou conjunto de objetos.

2.2.4.6 – XPointer

Linguagem que oferece uma sintaxe que permite localizar um recurso através da árvore de elementos do documento, isto é, . XPointer aponta para partes de documentos XML.

Uma vantagem do XPointer é que a mesma introduz Ponteiros Extendidos, que é um método sofisticado de localizar recursos. Em particular, os XPointers lhe permitem localizar recursos arbitrários em um documento, sem que seja necessário que o recurso seja identificado com um atributo ID.

2.2.4.7 – XLink

XLink é uma especificação W3C que permite definir vínculos entre documentos XML muito similar aos links HTML. XLink é uma coleção de atributos, com

namespace próprio, que podem ser usados em elementos de qualquer linguagem XML. As especificações *XLink* introduzem um modelo ligação padrão para a XML, que provê:

- Funcionalidade mínima é igual ao <a href> do HTML
- Funcionalidade estendida que permite vínculos bidirecionais, arcos, vários níveis de semântica, etc.

Para entender a diferença entre *XLink* e *XPointer*, pode-se afirmar que *XLink* é uma linguagem de construção de links e *XPointer* descreve como endereçar um recurso e como associar dois ou mais recursos.

2.2.4.8 – XML Schema

É uma linguagem para validação, especificação e definição de tipos de dados. O XML Schema consiste de três especificações:

- *Primer*: introdução;
- *Structures*: define estruturas (tipos complexos);
- *Datatypes*: define coleção de tipos simples.

Baseado nessa estrutura, XML Schema permite representar tipos simples (como texto, números, tokens, booleanos, etc.) e complexos (objetos, banco de dados, etc.), onde novos tipos podem restringir ou estender um tipo existente, ou serem criados a partir do nada..

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="professor">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nome" type="xs:string" />
        <xs:element name="foto">
          <xs:complexType>
            <xs:attribute name="href" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="endereco">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="logradouro" type="xs:string" />
              <xs:element name="bairro" type="xs:string" />
              <xs:element name="municipio" type="xs:string" />
              <xs:element name="uf" type="xs:string" />
            </xs:sequence>
            <xs:attribute name="tipo" type="xs:string" use="required" />
```

```
          </xs:complexType>
        </xs:element>
        <xs:element name="telefone">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="tipo" type="xs:string" use="required" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="email" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 11 – Um XML Schema baseado no XML da figura 1.

Para usar um XML *Schema* é preciso usar um *parser*[1] que o suporte. Um documento XML pode estar associado a vários documentos XML *Schema* - um por *namespace*.

É obrigatório o uso de XML *Schemas* para o suporte a outras tecnologias XML tais como *Web Services* (SOAP, WSDL, UDDI), XSLT 2.0, *XPath* 2.0, *XQuery*, entre outras.

2.2.4.9 – XHTML

XHTML (eXtensible HTML) é uma linguagem para formatação de XML com o objetivo de visualização, isto é, uma linguagem XML de descrição de página Web.

O XHTML tem os mesmos elementos do HTML 4.0 Strict, mas sua forma precisa ser especificada usando CSS (não há elementos/atributos para mudar cor, alinhamento, etc.), pois os elementos do XHTML descrevem somente a estrutura dos componentes da página.

Pode ser misturada (estendida) com outras linguagens XML (MathML, SVG, linguagens proprietárias).

2.2.4.10 – XML Namespaces

XML *Namespaces* ou espaço de nomes, serve para criar prefixos para os nomes de *tags*, evitando confusões que possam surgir com nomes iguais para *tags* que definem dados diferentes.

[1] *Parser* pode ser considerado um interpretador. MSXML 4.0 em diante é um ótimo *parser* XML.

XML *Namespaces* limita o escopo de elementos evitando conflitos quando duas ou mais linguagens se cruzam no mesmo documento e viabilizando a composição de documentos com vocabulários similares.

A sintaxe do XML *Namespace* consiste da associação de um identificador a cada elemento/atributo da linguagem, que pode ser:

- herdado através do escopo de uma sub-árvore;
- atribuído explicitamente através de um prefixo.

```
<?xml version="1.0" encoding="utf-8"?>
<professor xmlns:faculdade="FATEC">
  <nome>Liluyoud Cury de Lacerda</nome>
  <faculdade:nome>Faculdade de Tecnologia de Rondônia</faculdade:nome>
  <telefone tipo="Celular">(69) 3229-8969</telefone>
  <faculdade:telefone tipo="Celular">69 3229-8969</faculdade:telefone>
</professor>
```

Figura 12 – Exemplo de XML Namespaces

2.3 – Concluindo XML

Neste capítulo foram mostradas as principais características da XML, que é suficiente para entender os conceitos que serem abordados nessa dissertação como a relação entre banco de dados e XML (que será discutido no próximo capítulo e que servirá como base para a fundamentação dessa dissertação), o mapeamento de estruturas de dados para XML e vice-versa, e como esses conceitos podem contribuir para a comunicação e interação entre banco de dados heterogêneos.

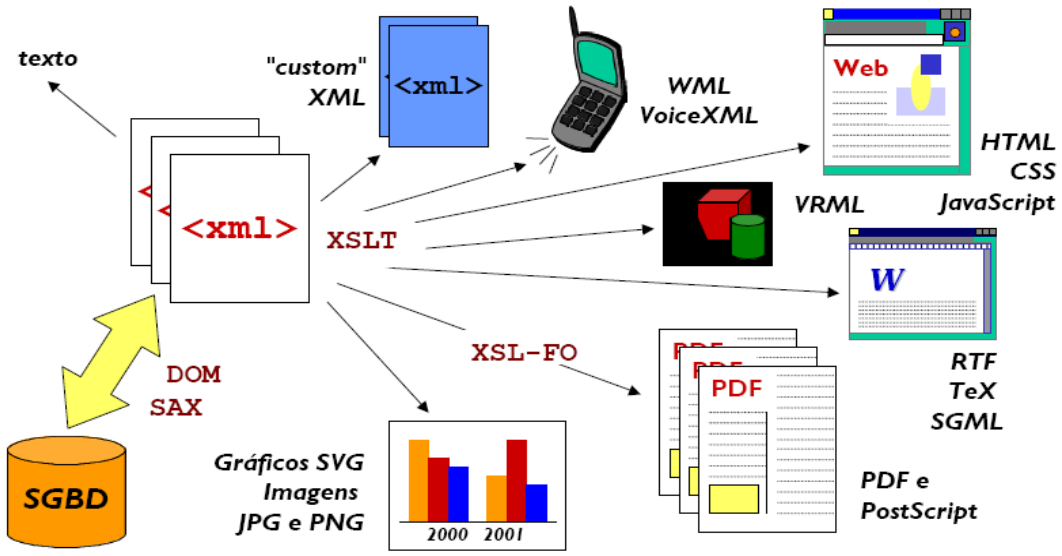


Figura 13 – O poder do XML.

Capítulo 3

XML E BANCO DE DADOS

Este capítulo discute as relações entre XML e banco de dados, e descreve alguns dos tipos de modelos disponíveis para processar documentos XML com banco de dados. Embora essa não seja uma análise muito detalhada, este capítulo aborda os assuntos mais importantes relacionados ao uso de XML com banco de dados[1].

Apesar de teoricamente a XML poder mapear qualquer tipo de informação, seja ela definida em estrutura hierárquica, relacional, orientada a objetos, entre outros, será focado somente os casos de estruturas relacionais, já que esse modelo é o dominante no meio.

3.1 – XML é um Banco de Dados?

Antes de ser discutido sobre XML e banco de dados, é interessante responder a uma questão muito comum: “XML é um Banco de Dados?”.

No sentido mais rígido, quando XML referencia um documento XML, a resposta é não. Embora um documento XML contenha dados sem um software adicional que ajude a processar estes dados, este documento não é mais banco de dados do que qualquer outro arquivo texto.

No sentido mais liberal, quando XML referencia um documento XML, e todas as ferramentas e tecnologia que cercam a XML, a resposta é sim, porém a resposta é sim porque XML prove muitas das características encontradas em um banco de dados, como:

- Armazenamento: o documento XML;
- Schemas: DTDs e linguagens de schema XML;

[1] Este estudo está um pouco direcionado para banco de dados relacionais por ser de longe, o mais usado.

- Linguagens de Consulta: XQL, XML-QL, QUILT, etc.1;
- Interfaces de programação: SAX (Simple API for XML ou API simples para XML) e DOM (Document Object Model ou Modelo de Objetos de Documentos)2, e assim por diante.

A XML pode apresentar essas características, mais ainda faltam alguns aspectos encontrados em banco de dados reais como: armazenamento eficiente, índices, segurança, transações e integridade de dados, acesso de múltiplos usuários, *triggers*[3], consultas ao longo de múltiplos documentos e assim por diante.

Então, enquanto é possível usar XML como um banco de dados em ambientes sem muitos dados com poucos usuários e modestos requerimentos de desempenho, ele poderá falhar na maioria de ambientes de produção, na qual têm muitos usuários, requerimentos de integridade de dados e que necessite de bom desempenho. Além disso, dado o baixo preço e a facilidade de usos de banco de dados como dBASE e Access parece haver pouca razão em usar XML até mesmo como um banco de dados em ambientes como o definido no primeiro caso acima.

Entretanto esses SGBDs têm padrões abertos, estão preparados para a web e são independentes de plataforma? E se integrar SGBD e XML para possuir o melhor das duas tecnologias? Será que essa integração traria benefícios para a aplicação em questão? É o que se pretende responder ao longo dessa dissertação.

3.2 - Porque usar um Banco de Dados?

A primeira questão que é necessário perguntar quando começa a pensar sobre XML e bancos de dados, é qual a finalidade de se usar um banco de dados em primeiro lugar. Tem dados legados que quer expor? Está procurando um lugar para armazenar suas páginas da Web? O banco de dados é usado por uma aplicação de comércio eletrônico na qual a XML é usado como um meio de transporte de dados? As respostas

[1] Maiores informações em *http://www.w3.org/TR/NOTE-xml-ql*
[2] Maiores informações em *http://www.javaworld.com/javaworld/jw-01-2000/jw-01-dbxml_p.html*
[3] Trigger (gatilho): ação disparada pelo banco de dados ao ser atualizado alguma informação.

para estas perguntas influenciarão fortemente sua escolha de banco de dados e middleware[1] (se houver), como também como é usado o banco de dados.

Por exemplo, suponha que uma aplicação de comércio eletrônico que use XML como meio de transporte de dados. Caso esses dados tenham uma estrutura altamente regular e que características como as entidades e codificações usadas em documentos XML não são importantes, já que o importante é somente os dados, e não como é armazenado fisicamente no documento, o uso da XML pode ser dispensável (em tese). Se sua aplicação for relativamente simples, um banco de dados relacional e um middleware de transferência de dados poderiam satisfazer plenamente as necessidades dessa aplicação, se for grande e complexo, seria altamente interessante haver um ambiente de desenvolvimento de aplicações completo que suporta XML.

Por outro lado, imagine um site na web constituídos de vários documentos XML (e de vários tipos), onde seria interessante prover um modo para usuários procurarem seus conteúdos, além de ferramentas para administrar o site. É provável que os documentos desse site tenham uma estrutura altamente irregular e coisas como uso de entidade é provavelmente importante porque eles são uma parte fundamental de como seus documentos são estruturados. Neste caso, poderia ser necessário algum tipo de "banco de dados XML nativo" que controla versões, rastros de uso de entidade, e suporte uma linguagem de consulta como XQL.

3.3 – Dados versus Documentos

Talvez o fator mais importante na escolha de um banco de dados é se o mesmo será usado para armazenar dados ou documentos.

Se for para armazenar dados, é necessário um banco de dados que seja afinado para armazenamento de dados, como um banco de dados relacional ou orientado a objeto, como também middleware para transferir dados entre o banco de dados e documentos de XML. Por outro lado, se for para armazenar documentos, será necessário um sistema de administração de conteúdo que é projetado especificamente para armazenar documentos.

[1] Aplicativo intermediário utilizado em muitos casos. Nesse especificamente seria um aplicativo que serviria como ponte entre a aplicação e o banco de dados.

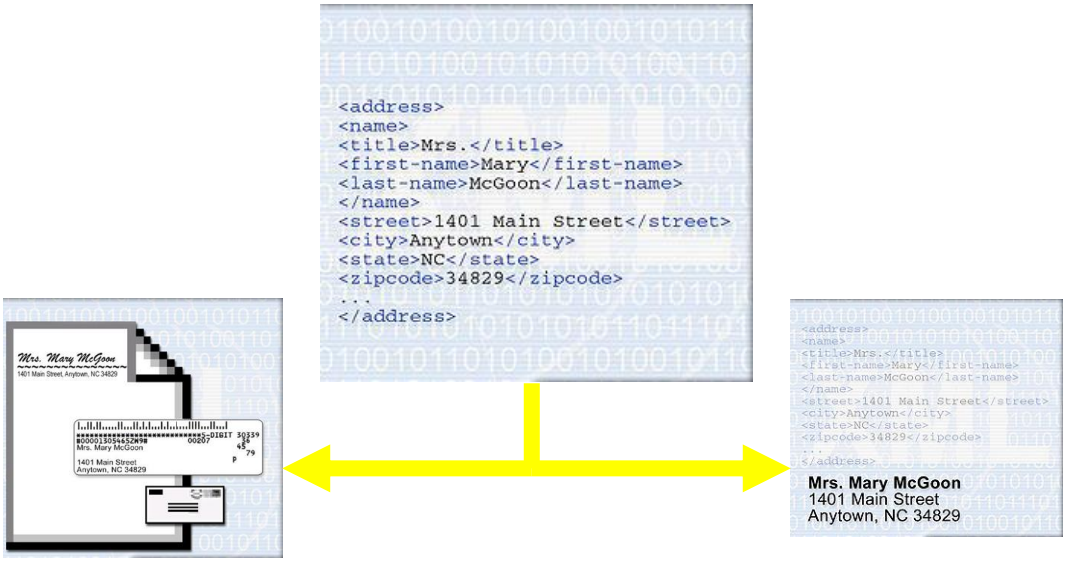


Figura 14 – Diferentes Visões dos mesmos dados

Embora seja possível armazenar documentos em um banco de dados relacional ou orientado a objeto, fatalmente teríamos um trabalho duplicado na implementação e implantação de um sistema de administração de conteúdo. Semelhantemente, embora um sistema de administração de conteúdo normalmente é construído em cima de um banco de dados orientado a objeto ou hierárquico, a tentativa de usá-lo como um banco de dados provavelmente será frustrante.

A necessidade de se armazenar dados ou documentos pode ser respondido simplesmente olhando para seus documentos de XML. A razão para isto é que a XML se concentra em duas categorias bem definidas: *centrado a dados* e *centrado a documentos* [SPD99].

3.3.1 – Documentos Centrados a Dados

Documentos centrados a dados são caracterizados através de uma estrutura bastante regular, dados bem organizados (quer dizer, a unidade independente menor de dados está ao nível de um único elemento *PCDATA* ou um atributo), e pequenos ou nenhum conteúdo misto. A ordem na quais elementos iguais e *PCDATA* acontecem não é freqüentemente significante. Exemplos bons disso é documentos XML contendo ordens de vendas, programa de vôo, cardápios de restaurante, e assim por diante.

Normalmente documentos centrados a dados são projetados para serem manipulados por máquinas, e o envolvimento do XML pode ser supérfluo, pois envolve somente transporte de dados. Entretanto essa abordagem será de grande valia, pois será usada para justificar a integração de banco de dados via XML, usando-o como o meio de comunicação onde os dados são transportados. Por exemplo, o documento de pedido de venda a seguir é centrado a dados:

```
<Venda VNum="12345">
  <Cliente CliNum="543">
    <CliNom>Indústrias ABC</CliNom>
    <Rua>Rua das Flores, 1313.</Rua>
    <Cidade>Florianópolis</Cidade>
    <Estado>SC</Estado>
    <CEP>60609</CEP>
  </Cliente>
  <DataPedido>20010101</DataPedido >
  <Linha LinNum="1">
    <Item IteNum="123">
      <Descricao>
        Roda Aro 14,Liga Leve, radial 185, Garantida de 6 meses.
      </Descricao>
      <Preco>109.95</Preco>
    </Item>
    <Quantidade>4</Quantidade>
  </Linha>
  <Linha LinNum="2">
    <Item IteNum="456">
      <Descricao>
        Laterna trazeira:Para Vectra 98 CD, garantia de um ano.
      </Descricao >
      <Preco>
        135.27
      </Preco >
    </Item>
    <Quantidade>2</Quantidade>
  </Linha>
</Venda>
```

Figura 15 – Exemplo de documento centrado a dados.

Muitos documentos ricos em prosa são na realidade centrados a dados, basta notar isso na XML mundial (web). Por exemplo, considere uma página na Amazon.com que exibe informação sobre um livro. Embora a página seja em grande parte texto, a estrutura daquele texto é altamente regular, onde muito deste texto é comum a todas as páginas que descrevem livros, e cada pedaço de texto específico da página está limitado em tamanho. Assim, a página poderia ser construída a partir de um documento XML simples centrado a dados, contendo texto específico que pode ser retornado de um banco de dados, e uma folha de estilo XSL que soma aos dados o texto de apresentação.

Em geral, a maioria dos sites da web que constrói dinamicamente documentos HTML preenchendo um modelo com dados de banco de dados, pode ser substituído provavelmente por documentos XML centrado a dados e um ou mais folhas de estilos - XSL. Por exemplo, observe o seguinte modelo de documento de arrendamento:

```
<Arrendamento>
  <Arrendatario>Industria ABC</Arrendatario> concorda em arrendar
  a propriedade na
  <Endereco>Rua das Flores, 1313, Florianópolis, SC </Endereco> do
  <Arrendador>Propriedades XYZ</Arrendador> para um termo de não
  menos que
  <Termo UnidadeTempo="Mês">18</Termo> a um custo de
  <Preco Moeda="R$" UnidadeTempo ="Mês">1000.00</Preco>
</Arrendamento>
```

Figura 16 – Exemplo de documento não centrado a dados.

Pode ser construído a partir de um documento XML centrado a dados mais uma simples folha de estilo, como mostra a figura a seguir:

```
<Arrendamento>
  <Arrendatario>Industria ABC</Arrendatario>
  <Endereco> Rua das Flores, 1313, Florianópolis, SC </Endereco>
  <Arrendador>Propriedades XYZ</Arrendador>
  <Termo UnidadeTempo="Mês">18</Termo>
  <Preco Moeda="R$" UnidadeTempo ="Mês">1000.00</Preco>
</Arrendamento>
```

Figura 17 – Exemplo de documento centrado a dados gerado a partir de um não centrado.

3.3.2 – Documentos Centrados a Documento

Documentos centrados a documento são caracterizados através de uma estrutura irregular, dados menos organizados quer dizer a menor unidade independente de dados poderia estar ao nível de um elemento com conteúdo misto ou o próprio documento inteiro, e muitos conteúdos misturados. A ordem nas quais elementos “iguais” e PCDATA acontecem quase sempre é significante. Exemplos bons são livros, e-mail, anúncios e quase qualquer documento de XHTML. Geralmente são projetados documentos centrados a documentos para consumo humano.

Por exemplo, o documento a seguir que descreve um produto, é centrado a documento:

```
<Produto>

  <Nome>Computador</Nome>

  <Fabricante>Indústrias Compaq</Fabricante>
```

```
  <Sumario>O estado da arte em harware</Sumario>

  <Descricao>

    <Para>
      o computador e modelo torre, com processador de 2GHz
      e memória RAM de 1Gb. Com velocidade de barramento de 400Mhz,
      Placa de vídeo de 64Mb RAM AGP 4x e Placa de som de 128 bits.
      Tem capacidade de armazenamento de 150Gb tecnologia Ultra3 SCSI
      e monitor de plasma 42 polegadas
    </Para>

    <Para>Você pode:</Para>

    <Lista>
      <Item>
        <Link URL="Pedido.html">Faca seu pedido agora </Link>
      </Item>
      <Item>
        <Link URL="conf.htm">Configurar o equipamento </Link>
      </Item>
      <Item>
        <Link URL="catalog.zip">Baixar catálogo</Link>
      </Item>
    </Lista>

    <Para>
      O frete é gratuito para qualquer lugar do Brasil, com
      previsão máxima para entrega de até 10 dias úteis. Se você fizer
      um pedido agora, ganhará uma impressora jato de tinta colorida
      da HP.
    </Para>

  </Descricao>
</Produto>
```

Figura 18 – Exemplo de documento centrado a documento.

3.3.3 – Dados, Documentos e Banco de Dados

Na realidade, a distinção entre documentos centrados a dados e centrados a documento não está sempre clara. Por exemplo, pode ocorrer o caso de um documento centrado a dados, como uma fatura conter dados irregularmente estruturados, como uma descrição de parte. E caso contrário, documento centrado a documento como um manual de um usuário poderia conter dados regularmente estruturados (freqüentemente metadados), como o nome de um autor e uma data de revisão. Apesar disto, caracterizando documentos como centrado a dados e centrado a documento, ajudará na avaliação de que se está interessado em dados ou documentos, que em troca ditam o tipo de sistema que deverá ser adotado.

Para armazenar e/ou retornar dados, pode-se usar um banco de dados (normalmente relacional, orientado a objeto ou hierárquico) e middleware (embutido ou

de terceiro), ou pode-se usar um servidor XML (uma plataforma para construir aplicações distribuídas, como comércio eletrônico que usa XML para transferência de dados) ou um servidor web habilitado para XML[1]. Para armazenar documentos, será necessário de um sistema de administração de conteúdo ou uma implementação de DOM persistente.

Serão discutidos esses assuntos, usando cada tipo de sistema, nas seções a seguir, "*Armazenando e Recuperando Dados*", e "*Armazenando e Recuperando Documentos*".

No Apêndice A, estão descritas uma lista de softwares disponíveis que integram banco de dados e XML.

3.4 – Armazenando e Recuperando Dados

Dados do tipo que são achados em documentos centrados a dados ou podem ter se originado no banco de dados (no caso de querer isto como XML) ou em um documento XML (no caso de querer armazenar isto em um banco de dados).

Um exemplo do primeiro caso, é a vasta quantidade de dados legado armazenada em bancos de dados relacionais (ou qualquer outro tipo de SGBD), no segundo exemplo é informação exposta na Web como XML que se quer armazenar em um banco de dados para processamento futuro.

Assim, dependendo das necessidades, pode-se precisar de software que transfere dados de um documento XML para o banco de dados, ou do banco de dados para um documento de XML, ou ambos[2].

3.4.1 – Transferindo Dados

Ao armazenar dados no banco de dados, é freqüentemente aceitável descartar muitas das informações sobre um documento, como seu nome e DTD, como também sua estrutura física como definição de entidade e uso, a ordem na qual valores de

[1] Um servidor web que pode construir documentos XML com dados a partir de um banco de dados.
[2] O software desenvolvido baseado nessa dissertação, o Simbah, tem o propósito de servir ambos os casos.

atributo e elementos acontecem, o modo no qual dados binários são armazenados (Base64 v. qualquer outra coisa), seções de CDATA, e informação codificada.

Semelhantemente, ao retornar dados do banco de dados, é provável que o documento de XML resultante contenha nenhum CDATA ou uso de entidade (diferente de entidades pré-definidas `lt`, `gt`, `amp`, `apos`, e `quot`) e é provável que a ordem no quais elementos e atributos aparece seja na ordem na qual os dados foram devolvidos pelo banco de dados.

Embora isto possa parecer estranho no princípio, o mesmo é bastante razoável. Por exemplo, considere o caso no qual XML é usado como um forma para transferir dados de uma ordem de vendas de um banco de dados para outro. Neste caso, não importa realmente se o número da encomenda da venda é armazenado no documento antes ou depois que a data da venda, nem importa se o nome do cliente é armazenado em uma seção de CDATA, uma entidade externa ou diretamente como PCDATA. O importante é que os dados pertinentes sejam transferidos de um banco de dados para outro (ou outros). Assim, o software de transferência de dados precisa considerar a ordem hierárquica (quais grupos de informação são sobre um único pedido de vendas) e outras coisas a mais.

Uma conseqüência de se ignorar informação sobre o documento e sua estrutura física é que, quando se armazena os dados de um documento no banco de dados e depois reconstrói o documento baseado naqueles dados armazenados, freqüentemente se resulta em um documento diferente. Se isto é aceitável depende de suas necessidades sendo que poderia influenciar na sua escolha de banco de dados e middleware de transferência de dados.

3.5 – Armazenado e Recuperando Documentos

Documentos tendem a originar arquivos XML ou algum outro formato, como RTF[1], PDF[2], ou SGML que então, são convertidos a XML. Assim, caso se trabalhe com

[1] *Rich Text Format*: Arquivo texto com formatação.
[2] Tipo de arquivo muito usado na web, implementado pela Adobe, na formatação de documentos.

documentos XML, provavelmente será necessário um modo para armazenar e recuperar esses documentos, como também modos para convertê-lo de e para outros formatos[1].

Para conjuntos de documentos simples, o sistema de arquivo ou algum tipo de sistema de controle de versão (como o usado para controle de versão de software) pode ser adequado para determinadas necessidades. Porém, se haver um conjunto de documentos complexos, certamente será necessário um sistema de administração de conteúdo[2].

Ao contrário do caso onde dados de um documento XML são armazenados no banco de dados, sistemas de administração de conteúdo geralmente suportam "*round-tripping* " de documentos, onde características relacionadas a estrutura física é muito crítica para manutenção dos documentos. Sistemas de administração de conteúdo também provêem várias outras capacidades, incluindo:

- Controle de acesso e versão;
- Ferramentas de pesquisa;
- Editores;
- Ferramentas de publicação em papel, CD, web, etc.;
- Separação de conteúdo e estilo;
- Extensibilidade por scripting ou programação;
- Integração de dados de banco de dados.

Como o objetivo dessa dissertação é mais sobre integração de banco de dados e não administração de conteúdo, não há necessidade de abordar mais detalhadamente

[1] Esta seção discutirá apenas alguns casos. Para mais informações sobre os softwares de conversão, seria interessante procurar em sites especializados em XML.

[2] O termo *sistema de administração de conteúdo*, ao invés de *sistema de gerenciamento de documentos*, reflete o fato que tais sistemas geralmente lhe permitem quebrar documentos em discretos fragmentos de conteúdo, como exemplos, procedimentos, capítulos, ou barras laterais, assim como também metadata, com nomes de autor, datas de revisão e números do documento, em lugar de ter que administrar cada documento como um todo. Isso não só simplifica tais coisas como coordenar o trabalho de múltiplos escritores que trabalham no mesmo documento, lhe permite compilar documentos completamente novos a partir de componentes existentes.)

sistemas de administração de conteúdo. Felizmente, tal detalhamento geralmente não é necessário, já que esses sistemas escondem a maioria de suas especificações técnicas.

3.5.1 – Sistemas de Administração de Conteúdo e Banco de Dados Relacionais

Pelo fato de sistemas de bancos de dados relacionais serem amplamente difundidos, a frase "armazenamento de documentos XML em um banco de dados" pode ser um sinônimo de sistemas de administração de conteúdo, e pode-se presumir que bancos de dados relacionais é uma boa maneira se armazenar documentos XML (ao invés dos dados nesses documentos). Se isto é uma boa idéia, certamente a resposta a essa pergunta ficará aberta[1].

Entretanto, a maioria das tentativas de se fazer isto não se concretizou de maneira satisfatória [AMA98]. A razão dada é que bancos de dados relacionais não estão preparados ainda para serem usados por sistemas de administração de conteúdo, pois o mesmo trata características como: ordem, hierarquia, estrutura irregular e campos de comprimento altamente variável.

Por exemplo, se for necessário armazenar informações sobre a ordem na qual PCDATA e elementos filhos aparecem em um elemento pai, é necessário armazenar isto em uma coluna separada e ordenar os filhos manualmente. Nem mesmo usando características estendidas do SQL, seria fácil formular a questão "adquira todos os capítulos nos quais o terceiro parágrafo menciona parte 123 em negrito".

Por outro lado, vários sistemas de administração de conteúdo, como *BladeRunner (Interleaf), SigmaLink (PASSO), Linguagem de Gerenciamento de Conteúdo (XyEnterprise)* e *Target2000 (Tecnologia de Informação Progressiva)*[2], estão baseados em bancos de dados relacionais. Um dos motivos citados é que bancos de dados relacionais estão mais maduros e disponíveis em maiores quantidades que bancos de dados orientados a objeto.

O que está claro é que um banco de dados relacional não é, em si mesmo, um sistema de administração de conteúdo. Quer dizer, se quiser fazer administração de conteúdo com um banco de dados relacional, será necessária a compra de um sistema de

[1] Depende dos objetivos de cada aplicação.
[2] Veja as especificações desses produtos no Apêndice A.

administração conteúdo baseado em um banco de dados relacional, aceitando as limitações de funcionalidade com uso de soluções de terceiros ou escrevendo um sistema próprio.

3.5.1.1 – Uso de Soluções de Terceiros

No pior caso, o uso de soluções de terceiros é limitado a armazenar documentos XML em formas não analisáveis e em uma única coluna como um BLOB[1]. Isto tem desvantagens óbvias, como a não capacidade de se criar um documento novo baseado nas partes de documentos existentes. Essa abordagem pode ser simples e funcional em alguns casos, como quando se tem uma descrição escrita em XHTML e não há necessidade de quebrá-lo em pedaços menores.

Além disso, bancos de dados relacionais estão continuamente expandindo suas capacidades de processamento, com a indexação de textos completos e a habilidade para executar procuras especializadas, como pesquisas por proximidade ou por sinônimos. Assim, armazenando documentos completos em uma única coluna pode ser suficiente para gerenciamento de documentos simples.

Felizmente, bancos de dados relacionais estão ficando mais atento a XML, assim uso de soluções de terceiros poderá armazenar documentos melhor do que colocá-los em uma única coluna. Embora que muitas das novas características XML são orientadas mais para transferência de dados do que gerenciamento de documentos, elas ajustam as necessidades de algumas aplicações, especialmente aquelas que envolvem transferência de dados e administração de conteúdo.

3.5.1.2 – Escrevendo um Sistema Próprio

Ao decidir escrever um sistema próprio, deve-se pensar muito antes de fazê-lo. Se realmente for necessário um sistema de administração de conteúdo com características complexas, pode-se estar desperdiçando um tempo precioso, pois tudo o que for feito pode já ter sido implementado por alguém. Por outro lado, se as necessidades são simples, escrever seu próprio sistema certamente dará valor ao seu tempo.

[1] Objeto binário muito grande: são geralmente imagens, vídeos, sons e etc.

Uma estratégia geral para escrever um sistema próprio é mapear o DOM (ou um subconjunto do DOM) para o banco de dados que usa um mapeamento objeto-relacional. Quer dizer, é criada uma tabela no banco de dados para cada objeto pertinente no DOM, e conectá-los usando chaves primárias e estrangeiras.

Como descrito, deve haver um mapeamento exato do DOM (ou aproximado dependendo de suas necessidades). Note que isto parece ser a estratégia usada por bancos de dados XML "nativos" construídos sobre bancos de dados relacionais.

Um exemplo deste tipo de sistema foi descrito por Mark Birbeck no fórum de discussão XML-L. O sistema consiste em cinco tabelas:

- *Definição de atributos:* define atributos, incluindo seus tipos, valores válidos, e assim por diante;
- *Associação de elemento/atributo:* define quais atributos são aplicados a quais elementos;
- *Definição do modelo de conteúdo:* define quais elementos podem conter outros elementos;
- *Valores de atributo:* contém valores de atributo e ponteiros para as linhas apropriadas na definição de atributo e tabelas de associação de elemento/atributo;
- *Valores de elemento:* contém valores de elementos (PCDATA ou ponteiros para outros valores de elemento), a ordem na qual o elemento ocorre em seu pai, um ponteiro para a linha que contém o valor do elemento pai, e um ponteiro para a linha apropriada na tabela de associação de elemento/atributo;

As primeiras três tabelas são equivalentes a um DTD simples; as outras duas tabelas contêm dados atuais. Pela contínua pesquisa dessas tabelas, é possível reconstruir qualquer parte de um documento de XML.

3.6 – Conclusões sobre Banco de Dados e XML

Pelo que foi exposto neste capítulo, foi visto que o relacionamento entre bancos de dados e XML está bastante madura (e ainda continua seu aperfeiçoamento) e isso é muito bom para garantir o uso de XML como padrão de intercâmbio de informações, se o que é pretendido é a interoperabilidade, transparência, simplicidade, padronização e principalmente compatibilidade com a internet.

Figura 19 – Modelo Relacional vs Modelo baseado em XML

No capítulo a seguir será descrito como se modela dados relacionais em XML, que complementará o que foi descrito nesse capítulo.

.

Capítulo 4

MODELANDO DADOS RELACIONAIS EM XML

Um banco de dados relacional[1] consiste em um conjunto de tabelas onde cada tabela é um conjunto de registros e um registro é um conjunto campos e cada campo é constituído pelo par: nome do campo e valor do campo. Todos os registros em uma determinada tabela têm o mesmo número de campos com um mesmo nome.

Este capítulo descreve como a XML (usando apenas um subconjunto da mesma para simplificar o entendimento) é usada para representar um banco de dados.

O modelo relacional também define certas restrições nas tabelas e define operações neles, mas os mesmos não serão levados em conta nesse capítulo. Em outras palavras, não se está tentando criar uma linguagem de consulta ou uma linguagem de definição de dados, e sim só uma linguagem que captura os dados em um banco de dados (uma visão particular do banco de dados), e colocá-los em um documento XML e vice-versa.

Muitas dessas linguagens são possíveis, sendo assim, não é difícil propor uma alternativa válida.

A descrição do banco de dados a seguir mostra um simples alinhamento de campos dentro de registros que estão dentro de uma tabela de um banco de dados. Este exemplo define uma ordem de compra, desde a sua modelagem até seu tratamento na XML.

[1] Este capítulo trata especificamente da modelagem de banco de dados relacionais ao invés de outros tipos de bases (como banco de dados orientado a objetos, hierárquico, etc.), pois os SGBDRs são os que estão mais presentes no mercado, além de apresentar tecnologia bem mais madura e definida que os outros modelos.

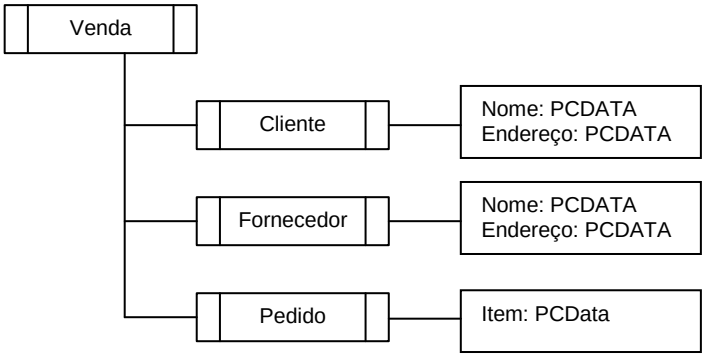


Figura 20 – Modelagem de uma Ordem de Compra - Estrutura

A partir dessa estrutura tem-se o seguinte documento XML:

```
<Venda>
  <Cliente>
    <Nome>Liluyoud Cury de Lacerda</Nome>
    <Endereco>Rua jamary, 597</Endereco>
  </Cliente>
  <Fornecedor>
    <Nome>UFSC</Nome>
    <Endereco>Florianopolis - SC</Endereco>
  </Fornecedor>
  <Pedido>
    <Item>Apostila de XML</Item>
  </Pedido>
</Venda>
```

Figura 21 – Modelagem de uma Ordem de Compra - XML

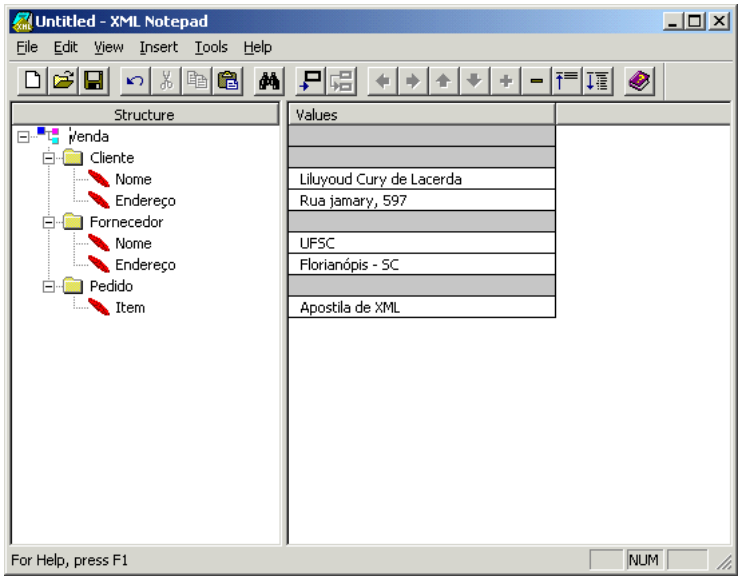


Figura 22 – Modelagem de uma Ordem de Compra – Ferramenta.

Observa-se que a XML é legível, podendo usar diversos tipos de ferramentas para visualização do conteúdo do banco de dados (parte ou todo), tais como navegadores e editores XML.

4.1 - Motivação

XML provê um ambiente completo para a integração de formas discrepantes de dados em uma plataforma de maneira independente. Desenvolvedores de todo o mundo estão trabalhando para expressar esses conceitos na sintaxe da XML.

Conceitos robustos de modelagem de informação foram desenvolvidos (e ainda estão) sobre a orientação do W3C[1] como objetivo de demonstrar a possibilidade do uso de qualquer modelo de dados em XML, através de conhecimentos básicos de conceitos e sintaxes de *schemas* XML, em particular dos DTDs[2]. Também se pretende abordar regras da XML que provê a criação de *schemas* automatizadas de fontes de dados relacionais.

4.2 - Elementos Básicos de Modelagem

4.2.1 – Hierarquia da modelagem

O banco de dados relacional pode também ser modelado como uma hierarquia de profundidade quatro [DM98]. A seguir serão definidas essas quatro profundidades que são o banco de dados, a tabela, o registro e o campo.

4.2.1.1 – Banco de Dados

Pode-se modelar um banco de dados usando o tipo elemento `<!ELEMENT>` ou atributo[3].

```
<!doctype name "url">
<nome>
  <tabela1>
  <tabela2>
   ......
```

[1] Coordenado pelo w3c schema working group.
[2] Vide Capítulo 2.
[3] Maiores informações em http://www.w3.org/xml/datamodel.html.

```
  <tabelan>
</nome>
```

Figura 23 – Exemplo de modelagem hierárquica do banco de dados em XML.

O `nome` é arbitrário, mas geralmente define o nome do banco de dados. A `url` é opcional, mas pode ser usado para apontar para informações sobre o banco de dados. No exemplo anterior não foi definida nenhuma `url`.

A ordem das tabelas também é arbitrária, desde que um banco de dados relacional não defina nenhuma ordenação nelas.

4.2.1.2 – Tabelas

Cada tabela de um banco de dados é representada pelo tipo elemento `<!ELEMENT>` nas quais os registros são seus filhos.

```
<nome>
  registro1
  registro2
  …
  registron
</nome>
```

Figura 24 – Exemplo de modelagem hierárquica de tabelas em XML.

O `<nome>` é o nome da tabela. A ordem dos registros é arbitrária, desde que o modelo de dados relacional não defina ordem a eles.

4.2.1.3 – Registro

Um registro também é representado por um tipo elemento `<!ELEMENT>`, nas quais seus filhos são os campos.

```
<nome>
  campo1
  campoo2
  …
  campon
</nome>
```

Figura 25 – Exemplo de modelagem hierárquica de registros em XML.

O `<nome>` é arbitrário, desde que o modelo de dados relacional não defina um nome para o tipo de registro. Porém, em XML o mesmo não pode ser omitido. Uma alternativa é re-usar o nome da tabela, ou, se a tabela tem um nome que é um plural, usar a forma singular (“pessoas” -> “pessoa”, “partes” -> “parte”).

A ordem dos campos é novamente sem importância.

4.2.1.4 – Campo

Um campo é representado por um tipo elemento `<!ELEMENT>` juntamente com um atributo como sendo seu único filho.

```
<name type="t">
  d
</name>
```

Figura 26 – Exemplo de modelagem hierárquica de campos em XML.

Se `d` for omitido, significa que o valor do campo é uma string vazia. O valor `t` indica o tipo do valor (como string, número, booleano, data[1]). Se o tipo de atributo for omitido, o tipo é assumido como sendo string.

4.2.1.4.1 – Valores nulos

Valores nulos são representados pela ausência do campo. Note que isto é diferente de deixar o campo vazio que indica que o campo contém uma string de comprimento zero. Valores nulos têm propriedades especiais em bancos de dados relacionais. Por exemplo, dois campos ambos com valores nulos não são iguais (em contraste com dois campos com string de tamanho zero, que são).

4.2.2 – Forte Tipificação

Tim Bray[2] escreveu uma proposta para acrescentar forte tipificação a XML, usando um conjunto de atributos fixos. O exemplo anterior adquiriria atributos declarados como visto abaixo:

```
<livro>

  <autores>
    <?xml default nome
    xml-sqltype="varchar"
    xml-sqlsize="40"?>

    <?xml default endereco
    xml-sqltype="varchar"
    xml-sqlsize="40"?>
    ...
    <?xml default nascimento
    xml-sqltype="data"
    xml-sqlmin="1900/01/01"
```

[1] A lista completa de tipos pode variar de uma implementação para outra, sendo que um padrão está definido pela W3C.
[2] Documento encontrado em http://www.textuality.com/xml/typing.html

```
    xml-sqlmax="1990/01/01"?>
    ...
  </autores>

  <editores>
    <?xml default nome
    xml-sqltype="varchar"
    xml-sqlsize="40"?>
    ...
  </editores>

</livro>
```

Figura 27 – Exemplo de forte tipificação aplicado a XML.

Isto permitirá que uma aplicação saiba sobre estes atributos para conferir o conteúdo de cada campo.

4.3 – Abordagens para Modelagem

Ao mapear informação de modelos de dados tradicionais (dados hierárquico, relacional e orientado a objeto) para XML, não há uma única resposta certa. Ao invés disso, existem várias abordagens potenciais [SPD99].

Um tipo de abordagem é mapear pedaços individuais de informação do banco de dados em um *schema* "pré-ordenado" (por exemplo: um formato de intercâmbio padrão de mercado). Esta aproximação implica em códigos padronizados para executar os mapeamentos entre o dois *schemas* (um do banco de dados e outro do schema pré-ordenado).

Outro tipo de abordagem é o "*data-dump*"[1] de todo o conteúdo de um banco de dados com a intenção de re-constituir todo o banco de dados com todas as suas relações e dados intactos. Esta abordagem tende a render um mapeamento mecânico sobre um *schema* genérico de metadados.

Ambas abordagens são modos comuns e úteis de modelar dados relacionais. Ambos têm suas desvantagens: a necessidade de código padrão significativo, e o obscurecimento de conceitos de negócios.

Neste capítulo será usado uma abordagem que está entre este dois extremos e aplicá-los a bancos de dados relacionais. Essa abordagem provê mapeamento mecânico

[1] Recuperar informações como um todo, sem haver uma pré-análise

dos conceitos fundamentais da estrutura do banco de dados enquanto se guarda a noção de como o *schema* deveria saber sobre as informações do negócio. Esta aproximação tem o mérito de gerar *schemas* facilmente compreensíveis que correspondente com as estruturas de dados existentes, usando conceitos XML como *ID/IDREF*.

Esta aproximação não garante prover 100% de fidelidade ao trocar dados de um SGBDR para XML, e deste de volta ao SGBDR, mas sempre que possível, será identificado tais limitações e soluções alternativas em potencial.

4.3.1 – Um Exemplo Simples

Será usado um exemplo simples para trabalhar nele. O banco de dados usado será o de empregados (*EMPREGADO*), onde tem três tabelas.

EMPREGADO: que armazena informações pessoais.
REVISAO_PERFIL: que armazena informações de desempenho sobre empregado.
MUDANCA_SALARIAL: que armazena mudanças do salário do empregado resultado (normalmente, mas não sempre) de uma revisão de desempenho do mesmo.

```
   TABLE EMPREGADO
      NUM LONGINT PRIMARY KEY
      PNOME STRING 32
      UNOME STRING 32
      DATA_CONTRATACAO DATE
      DATA_SAIDA DATE MAY BE NULL
   TABLE REVISAO_PERFIL
      EMP_NUM LONGINT PRIMARY KEY FOREIGN KEY
      DATA_REVISAO DATE PRIMARY KEY
      REVISAO TEXT
   TABLE MUDANCA_SALARIAL
      EMP_NUM LONGINT FOREIGN KEY
      DATA_REVISAO DATE MAY BE NULL
      DATA_MUDANCA DATE
      SALARIO INT
```

Figura 28 – Definição do banco de dados de empregados.

Um documento XML de informação tirado deste exemplo poderia se parecer assim:

```
<EMPREGADO
      NUM = '2361'
      PNOME = 'Sara'
      UNOME = 'Luize'
      DATA_CONTRATACAO = '4/4/98'>
</EMPREGADO>
```

Figura 29 – Exemplo de um documento do banco de dados de empregados.

Note que um elemento nomeado depois da tabela corresponde a um registro de dados com atributos que contêm os valores para cada coluna. A partir disso, existem melhorias cruciais que se deve fazer ao modelo:

1. Suporte a informações dos tipos de dados;
2. Definição das relações fundamentais
3. Uso das facilidades *ID/IDREF* do XML

Para fazer estas melhorias serão necessárias informações adicionais no *schema* usando técnicas de esboço.

4.3.2 – Estendendo o DTD

A sintaxe de DTD é muito eficiente para expressar certos conjuntos de restrições em determinados documentos. Por isso o DTD será estendido para capturar tal informação adicional para os *Tipos de Elementos* e *Tipos de Atributo* que será definido em nosso *schema*.

Serão usados atributos fixos para realizar isto porque o valor dos atributos fixos é inalterável em instâncias individuais de um elemento, por isso eles podem ser pensados como propriedades de um tipo de elemento em lugar de uma instância. Assim:

- Para tipos de elemento associa-se a propriedade adicionando um atributo fixo `nmtoken`, cujo nome é precedido por um `e-`. O valor deste atributo é o valor da propriedade. Por exemplo, para associar `'vermelho'` como o valor da propriedade `'cor'` para o tipo de elemento `'retangulo'`, seria necessário definir um atributo `e-cor` cujo valor fixo fosse `'vermelho'`.

- Para tipos de atributo associa-se a propriedade adicionando o par nome de tipo de atributo e valor da propriedade, para um atributo fixo `nmtokens` cujo nome é o nome de propriedade precedido por um `'a-'`. Por exemplo, para associar `'vermelho'` como o valor da propriedade `'cor'` para um tipo de atributo particular chamado `'retangulo'`, será necessário definir um atributo `a-cor` cujo valor fixo seria `'azul'` contido em `'retangulo azul'`.

```
<!ATTLIST retangulo azul CDATA #IMPLIED
   tamanho CDATA #IMPLIED
   e-cor NMTOKEN #FIXED 'azul'
   a-cor NMTOKENS #FIXED 'retangulo azul'
>
```

Figura 30 – Exemplo de um documento do banco de dados de empregados.

Para atingir os propósitos precisar-se-á de mais quatro propriedades:

- *dtype:* Tipo de dado para um tipo de elemento ou atributo. É usado para especificar os valores permitidos para um elemento do tipo PCDATA ou um atributo de tipo CDATA. Para uma discussão do uso e significado da propriedade, veja a seção "Modelando Tipos de Dados" mais adiante.

- *dsize:* Indica o tamanho de armazenamento necessário para um tipo de dados.

- *pkey:* O nome do atributo ou tipos de elementos que correspondem a uma coluna de chave primária. No caso de chaves agregadas, é definida uma lista delimitada por espaços. Isto só é usado no tipo de elemento que corresponde a uma tabela.

- *fkey:* É uma referência para o atributo ou tipo de elemento que representam as colunas na qual este atributo ou tipo de elemento se refere. É usado somente no caso do atributo ou tipo de elemento for relacionado a uma coluna que contém uma chave estrangeira.

4.3.3 – Modelando Tipos de Dados

Fontes de dados tradicionais tendem a ser fortemente tipificadas. Enquanto um documento XML, por definição, apresenta normalmente sua informação em forma textual, escondendo conhecimentos sobre o tipo de dados de seus elementos e atributos.

A propriedade *dtype* de metadados (como esboçado acima) será usada para capturar informações do tipo de dados. Que conjunto de valores de tipos de dados podem ser usados? Enquanto uma lista definitiva de tipos de dados é um objetivo ainda

evasivo (se não ilusório), um conjunto útil foi compilado pelas submissões W3C[1]. Será usada nesse trabalho essa convenção de definição de nomes de tipos de dados.

String	number	dateTime
Boolean	Float	date
Uri	Int	time

Figura 31 – Tabela com os tipos de dados definidos pela W3C.

Um problema relacionado surge sobre o tamanho de armazenamento do tipo de dado. Por exemplo, quantos bytes serão necessários para armazenar um valor do tipo *"int"* ou qual o tamanho máximo do caractere que compõe o tipo *"string"*? E até mesmo, qual a precisão do valor do tipo *"number"*?. Para tanto é usada a propriedade *dsize* para definir esta informação. Estes são os valores atribuídos ao *dsize*:

dtype	***Dsize***	***Example***
String	Número máximo de caracteres	23
Float	De 4 a 8 bytes	4
Number	x.y onde: x = dígitos permitidos a esquerda do decimal y = dígitos permitidos a direita do decimal	14.4
Int	Pode ser de 1, 2, 4 ou 8 bytes i1, i2, i4, i8 podem ser usados no lugar de int abreviar o atributo dsize	8

Figura 32 – Tabela com tamanhos dos dados definidos pela W3C.

Note que esta codificação de informação está dentro da informação do tipo de dado dentro do DTD, e que a validação dessas informações não são feitas por si só. Tal validação é atribuída aos analisadores (*parsers*) XML. Usando estes métodos, podem-se disponibilizar tais informações para seu código para validações apropriadas. Comparando essas notações com os valores definidos no *dtype*, pode-se atribuir uma referência aos módulos de validação de sua aplicação, uma para cada tipo de dado.

[1] Muitos dos padrões do w3c foram definidas a partir de RFC (Request For Comments ou Requisitando Para Comentários).

4.3.4 – Modelando Relacionamentos

Muito do poder e complexidade de mapear dados para XML vem do mapeamento das relações entre os pedaços de dados. No mundo relacional isto é comparado a modelar os relacionamentos de chave primária e estrangeira. A seguir serão comentados esses relacionamentos.

4.3.4.1 – Chaves Primárias

Uma chave primária provê um único valor pelo qual um único registro de uma tabela pode ser acessada. XML tem um conceito semelhante baseado no atributo *ID* que provê acesso único a um elemento.

Muitas implementações DOM provê acesso indexado a tais elementos e que são freqüentemente desejáveis. Porém, surge um problema: *IDs* XML tem que ser únicos em todo o documento, e uma chave primária só é única dentro daquela coluna.

Para prover a singularidade global necessária, é usado um atributo *pkey_id* suplementar para guardar uma versão dos dados da chave primária que foi feito globalmente único. Para fazer isto, tira-se proveito de dois fatos:

- A chave primária é sem igual dentro do contexto do tipo de elemento que contém cada registro (nomeado depois da própria tabela), e
- O nome do tipo de elemento é único dentro do documento. Assim, para fazer o nome único localmente ser único globalmente, é associado o nome do elemento com o valor da chave.

Por exemplo:

```
<EMPREGADO pkey_id = "EMPREGADO.2361">
  <EMPREGADO.NUM>2361</EMPREGADO.NUM>
  <EMPREGADO.PNOME>Liluyoud</EMPREGADO.PNOME>
  <EMPREGADO.UNOME>Lacerda</EMPREGADO.UNOME>
  <EMPREGADO.DATA_CONTRATACAO>4/4/88</EMPREGADO.DATA_CONTRATACAO>
</EMPREGADO>
```

Figura 33 – Definindo chaves primárias em um documento XML.

4.3.4.2 – Chaves Estrangeiras

A presença de chaves estrangeiras dentro de tabela provê um relacionamento entre tabelas diferentes. Da mesma maneira que o *ID* no XML prove o conceito de

chaves primárias, a noção de *IDREF* provê um conceito análogo para chaves estrangeiras.

Para acomodar o conceito de chave estrangeira ao XML, é usada uma técnica semelhante ao da chave primária. É criado um novo atributo *IDREF* para o elemento de tabela cujo nome é derivado do nome da coluna (com um *_idref* junto).

Valores em um documento serão construídos de uma maneira similar para os atributos *pkey_id* (realmente a produção tem coincidir precisamente, caso contrário, todos os ponteiros estarão perdidos).

Por exemplo:

```
<EMPREGADO pkey_id = "EMPREGADO.2361">
  <EMPREGADO.NUM>2361</EMPREGADO.NUM>
  ...
<REVISAO_PERFIL REVISAO_PERFIL.EMP_NUM_idref = "EMPREGADO.2361">
  <REVISAO_PERFIL.EMP_NUM>2361</REVISAO_PERFIL.EMP_NUM>
    ...
```

Figura 34 – Definindo chaves estrangeiras em um documento XML.

Outro problema que surge é a necessidade para ter o *schema* – ao invés de somente um documento – que descreva os relacionamentos envolvidos. Note isso no exemplo acima, que os dados do elemento *REVISAO_PERFIL*, que associa *REVISAO_PERFIL* com EMPREGADO. Dessa maneira o *schema* deve "saber" do relacionamento na ausência de uma instância do documento, definindo aquele relacionamento usando as propriedades *pkey* e *fkey*.

Por exemplo:

```
<!ATTLIST EMPREGADO e-pkey NMTOKEN #FIXED 'EMPREGADO.NUM'>

e

<!ATTLIST REVISAO_PERFIL.EMP_NUM e-fkey NMTOKEN #FIXED
'EMPREGADO.EMPREGADO_NUM'>
```

Figura 35 – Definindo chaves estrangeiras em um schema XML.

4.3.4.3 – Aninhamentos

Há uma alternativa ao mecanismo descrito acima para alguns casos. Esta técnica tira proveito do fato que o relacionamento em XML pode ser deduzido do contexto de hierarquia como também por relações de *id/idref*. Assim, em nosso exemplo, pode-se

associar elementos *REVISAO_PERFIL* com os seus elementos correspondentes de *EMPREGADO* os colocando aninhados um a outro.

Por exemplo:

```
<EMPREGADO EMPREGADO.NUM_id = "EMPREGADO.2361"
      NUM = '2361'
      PNOME = 'Liluyoud'
      UNOME = 'Lacerda'
      DATA_CONTRATACAO = '4/4/88'>
  <REVISAO_PERFIL
     DATA_REVISAO = '1/1/98'
     REVISAO = 'Eficiente'/>
  <REVISAO_PERFIL
     DATA_REVISAO = '1/1/99'
     REVISAO = 'Alta performance'/>
</EMPREGADO>
```

Figura 36 – Definindo relacionamentos com técnicas de aninhamento.

Note que isto nem sempre é desejável, e freqüentemente, nem é mesmo possível. As seguintes condições devem ser conhecidas para isto ser possível:

- A chave estrangeira não deve ser nula (nem opcional);
- Deve ser a única chave estrangeira modelada na tabela;
- Toda linha (registro) desejada tem que referenciar para uma linha que será incluída;
- A chave estrangeira não deve apontar para a mesma tabela.

4.4 – Um Exemplo Passo a Passo

4.4.1 – Modelando Tabelas

Tabela: A própria tabela se torna um tipo de elemento. No documento cada ocorrência de um elemento deste tipo conterá uma única linha de dados.

Se estiver modelando as colunas desta tabela como atributos, o conteúdo do modelo dever estar vazio (*EMPTY*) se estiver modelando as colunas como elementos, o conteúdo do modelo é uma sucessão de elementos que correspondem às colunas. No segundo caso deve ser definido opcional (?) qualquer elemento cujos valores das colunas podem ser nulos.

```
<!ELEMENT EMPREGADO EMPTY>

ou

<!ELEMENT EMPREGADO (
  EMPREGADO.NUM,
  EMPREGADO.PNOME,
  EMPREGADO.UNOME,
  EMPREGADO.DATA_CONTRATACAO,
  EMPREGADO.DATA_SAIDA)>
```

Figura 37 –Modelando uma tabela.

1. *Chave primária.* Se a tabela tem uma chave primária, essa característica é definida com a propriedade *pkey*.

```
<!ATTLIST EMPREGADO e-pkey NMTOKEN #FIXED 'EMPREGAO.NUM'>
```

Figura 38 – Modelando chave primária.

2. *Chave estrangeira:* Para prover acesso *id/idref* a relacionamentos de chaves primárias e estrangeiras, cria-se um atributo de *ID* especial para conter uma versão única global desta chave para cada instância do elemento.

```
<!ATTLIST EMPREGADO pkey_id ID #REQUIRED>
```

Figura 39 – Modelando chave estrangeira.

4.4.2 – Modelando Colunas como Elementos

Para cada coluna é necessário:

1. *Nome de coluna:* É um tipo de elemento com o mesmo nome da coluna original. No documento, cada ocorrência de um elemento deste tipo conterá o valor de uma única coluna de um único registro de dados.

   Já que os nomes dos tipos de elemento devem ser únicos ao longo de todo o documento, deve-se qualificar o nome de tipo de elemento da coluna com o nome da tabela a qual pertence.

```
<!ELEMENT EMPREGADO.DATA_SAIDA (#PCDATA)>
```

Figura 40 – Modelando colunas de uma tabela como elementos

2. *Tipo de dados de coluna:* tipos de dados são definidos usando a propriedade dtype, já discutido anteriormente.

```
<!ATTLIST EMPREGADO.DATA_SAIDA e-dtype NMTOKEN #FIXED 'date'>
```

Figura 41 – Modelando os tipos de dados das colunas de uma tabela como elementos

3. *Colunas nulas:* Como já visto, se a coluna pode conter um valor nulo então sua referência no modelo de conteúdo de tabela é opcional.

4. *Coluna com Chave Estrangeira:* Se a coluna contém uma chave estrangeira, é usada a propriedade *fkey* para registrar o elemento da coluna ao qual a chave se refere.

```
<!ATTLIST REVISAO_PERFIL.EMP_NUM e-fkey NMTOKEN #FIXED
'EMPREGADO.NUM'>
```

Figura 42 – Modelando chaves estrangeiras como elementos

5. Para prover acesso *id/idref* para relacionamentos de chaves primárias e estrangeiras, um atributo tipo elemento da tabela pode ser criado para servir como uma ligação de *IDREF* ao elemento correspondente a outra tabela. O nome de atributo é formado juntando *"_idref"* ao nome do elemento de coluna.

```
<!ATTLIST REVISAO_PERFIL REVISAO_PERFIL.EMP_NUM_idref IDREF #REQUIRED>
```

Figura 43 – Nomeando chaves estrangeiras como elementos

4.4.3 – Modelando Colunas como Atributos

Para cada coluna é necessário:

1. *Nome de coluna:* Um novo tipo de atributo com o mesmo nome é criado. No documento, cada ocorrência de um atributo deste tipo conterá o valor de uma única coluna de um único registro de dados.

```
<!ATTLIST EMPREGADO EMPREGADO.DATA_SAIDA CDATA #IMPLIED>
```

Figura 44 – Modelando colunas de uma tabela como Atributos.

2. *Tipo de dados de coluna:* Tipos de dados são definidos usando a propriedade *dtype*.

```
<!ATTLIST EMPREGADO
  a-dtype NMTOKENS #FIXED
    'EMPREGADO.NUM int
     EMPREGADO.PNOME string
     EMPREGADO.UNOME string
     EMPREGADO.DATA_CONTRATACAO date
     EMPREGADO.DATA_SAIDA date'
  a-dsize NMTOKENS #FIXED
    'EMPREGADO.PNOME 32
     EMPREGADO.UNOME 32'>
```

Figura 45 – Modelando os tipos de dados das colunas de uma tabela como atributos

3. *Coluna Nula:* Se a coluna pode conter um valor nulo então de atributo é implícito, se não, é exigido.

4. *Coluna com Chave Estrangeira:* Se a coluna contém uma chave estrangeira, é usada a propriedade *fkey* para registrar o elemento da coluna ao qual a chave se refere.

```
<!ATTLIST REVISAO_PERFIL a-fkey NMTOKENS #FIXED 'EMP_NUM
EMPREGADO.NUM' >
```

Figura 46 – Modelando chaves estrangeiras como atributos

5. Para prover acesso *id/idref* para relacionamentos de chaves primárias e estrangeiras, um atributo tipo elemento da tabela pode ser criado para servir como uma ligação *IDREF* ao elemento correspondente a outra tabela. O

nome de atributo é formado juntando *"_idref"* ao nome do elemento de coluna.

```
<!ATTLIST REVISAO_PERFIL EMP_NUM_idref IDREF #REQUIRED>
```

Figura 47 – Nomeando chaves estrangeiras como atributos

4.4.4 – Listagem Completa do Exemplo

4.4.4.1 – DTD com Colunas Baseado em Elementos

```
<!ELEMENT EMPREGADO (
   EMPREGADO.NUM?,
   EMPREGADO.PNOME,
   EMPREGADO.UNOME,
   EMPREGADO.DATA_CONTRATACAO,
   EMPREGADO.DATA_SAIDA?)>
<!ATTLIST EMPREGADO pkey_id ID #REQUIRED e-pkey NMTOKEN #FIXED 'EMPREGADO.NUM' >

<!ELEMENT EMPREGADO.NUM (#PCDATA )>
<!ATTLIST EMPREGADO.NUM e-dtype NMTOKEN #FIXED 'int' >

<!ELEMENT EMPREGADO.PNOME (#PCDATA )>
<!ATTLIST EMPREGADO.PNOME e-dtype NMTOKEN #FIXED 'string' e-dSize NMTOKEN #FIXED '32' >

<!ELEMENT EMPREGADO.UNOME (#PCDATA )>
<!ATTLIST EMPREGADO.UNOME e-dtype NMTOKEN #FIXED 'string' e-dSize NMTOKEN #FIXED '32' >

<!ELEMENT EMPREGADO.DATA_CONTRATACAO (#PCDATA )>
<!ATTLIST EMPREGADO.DATA_CONTRATACAO e-dtype NMTOKEN #FIXED 'date' >

<!ELEMENT EMPREGADO.DATA_SAIDA (#PCDATA )>
<!ATTLIST EMPREGADO.DATA_SAIDA e-dtype NMTOKEN #FIXED 'date' >

<!ELEMENT REVISAO_PERFIL (
   REVISAO_PERFIL.EMP_NUM ,
   REVISAO_PERFIL.DATA_REVISAO ,
   REVISAO_PERFIL.REVISAO )>
<!ATTLIST REVISAO_PERFIL REVISAO_PERFIL.EMP_NUM_idref IDREF #REQUIRED >

<!ELEMENT REVISAO_PERFIL.EMP_NUM (#PCDATA )>
<!ATTLIST REVISAO_PERFIL.EMP_NUM e-dtype NMTOKEN #FIXED 'int' e-fkey NMTOKEN #FIXED
'EMPREGADO.NUM' >

<!ELEMENT REVISAO_PERFIL.DATA_REVISAO (#PCDATA )>
<!ATTLIST REVISAO_PERFIL.DATA_REVISAO e-dtype NMTOKEN #FIXED 'date' >

<!ELEMENT REVISAO_PERFIL.REVISAO (#PCDATA )>
<!ATTLIST REVISAO_PERFIL.REVISAO e-dtype NMTOKEN #FIXED 'string' e-dSize NMTOKEN #FIXED
'50' >

<!ELEMENT MUDANCA_SALARIAL (
   MUDANCA_SALARIAL.EMP_NUM ,
   MUDANCA_SALARIAL.DATA_REVISAO? ,
   MUDANCA_SALARIAL.DATA_MUDANCA ,
   MUDANCA_SALARIAL.SALARIO )>
<!ATTLIST MUDANCA_SALARIAL MUDANCA_SALARIAL.EMP_NUM_idref IDREF #REQUIRED >

<!ELEMENT MUDANCA_SALARIAL.EMP_NUM (#PCDATA )>
<!ATTLIST MUDANCA_SALARIAL.EMP_NUM e-dtype NMTOKEN #FIXED 'int' e-fkey NMTOKEN #FIXED
'EMPLOYEE.NUM' >

<!ELEMENT MUDANCA_SALARIAL.DATA_REVISAO (#PCDATA )>
<!ATTLIST MUDANCA_SALARIAL.DATA_REVISAO e-dtype NMTOKEN #FIXED 'date' >
```

```
<!ELEMENT MUDANCA_SALARIAL.DATA_MUDANCA (#PCDATA )>
<!ATTLIST MUDANCA_SALARIAL.DATA_MUDANCA e-dtype NMTOKEN #FIXED 'date' >

<!ELEMENT MUDANCA_SALARIAL.SALARIO (#PCDATA )>
<!ATTLIST MUDANCA_SALARIAL.SALARIO e-dtype NMTOKEN #FIXED 'int' >
```

4.4.4.2 – DTD com Colunas Baseado em Atributos

```
<!ELEMENT EMPREGADO EMPTY>
<!ATTLIST EMPREGADO pkey_id ID #REQUIRED
   NUM CDATA #REQUIRED
   PNOME CDATA #REQUIRED
   UNOME CDATA #REQUIRED
   DATA_CONTRATACAO CDATA #REQUIRED
   DATA_SAIDA CDATA #IMPLIED
   e-pkey NMTOKEN #FIXED 'NUM'
   a-dtype NMTOKENS 'NUM int
                     PNOME string
                     UNOME string
                     DATA_CONTRATACAO date
                     DATA_SAIDA date'
   a-dSize NMTOKENS 'PNOME 32 UNOME 32' >

<!ELEMENT REVISAO_PERFIL EMPTY>
<!ATTLIST REVISAO_PERFIL REVISAO_PERFIL.EMP_NUM_idref IDREF #REQUIRED
   EMP_NUM CDATA #REQUIRED
   DATA_REVISAO CDATA #REQUIRED
   REVISAO CDATA #REQUIRED
   a-dtype NMTOKENS 'EMP_NUM int
   DATA_REVISAO date
   REVISAO date'
   a-fkey NMTOKENS 'EMP_NUM EMPREGADO.NUM'
   a-dSize NMTOKENS 'REVISAO 50' >

<!ELEMENT MUDANCA_SALARIAL EMPTY>
<!ATTLIST MUDANCA_SALARIAL MUDANCA_SALARIAL.EMP_NUM_idref IDREF #REQUIRED
   EMP_NUM CDATA #REQUIRED
   DATA_REVISAO CDATA #IMPLIED
   DATA_MUDANCA CDATA #REQUIRED
   SALARIO CDATA #REQUIRED
   a-dtype NMTOKENS 'EMP_NUM int
   DATA_REVISAO date
   DATA MUDANCA date
   SALARIO int'
   a-fkey NMTOKENS 'EMP_NUM EMPREGADO.NUM' >
```

4.4.4.3 – Dados XML com Colunas Baseadas em Elemento

```
<?xml version ="1.0"?>
<!--Generated by XML Authority. Conforms to XML Data subset for IE 5-->
<Schema name = ""
   xmlns = "urn:schemas-microsoft-com:xml-data"
   xmlns:dt = "urn:schemas-microsoft-com:datatypes"
   xmlns:xa = "www.extensibility.com/schemas/xdr/metaprops.xdr">

<ElementType name = "EMPREGADO" xa:pkey = "EMPREGADO.NUM" content = "eltOnly" order =
"seq">
   <AttributeType name = "pkey_id" dt:type = "ID" required = "yes"/>
   <attribute type = "pkey_id"/>
   <element type = "EMPREGADO.NUM" />
   <element type = "EMPREGADO.PNOME" />
   <element type = "EMPREGADO.UNOME" />
   <element type = "EMPREGADO.DATA_CONTRATACAO />
   <element type = "EMPREGADO.DATA_SAIDA" minOccurs = "0" maxOccurs = "1"/>
</ElementType>

<ElementType name = "EMPREGADO.NUM" content = "textOnly" dt:type = "i4"/>
<ElementType name = "EMPREGADO.PNAOE" content = "textOnly" dt:type = "string"/>
<ElementType name = "EMPREGADO.UNOME" content = "textOnly" dt:type = "string"/>
<ElementType name = "EMPREGADO.DATA_CONTRATACAO" content = "textOnly" dt:type = "date"/>
<ElementType name = "EMPREGADO.DATA_SAIDA" content = "textOnly" dt:type = "date"/>
```

```
<ElementType name = "REVISAO_PERFIL" content = "eltOnly" order = "seq">
   <AttributeType name = " REVISAO_PERFIL.EMP_NUM_idref" dt:type = "IDREF" required =
"yes"/>
   <attribute type = "REVISAO_PERFIL.EMP_NUM_idref"/>
   <element type = "REVISAO_PERFIL.EMP_NUM" />
   <element type = "REVISAO_PERFIL.DATA_REVISAO />
   <element type = "REVISAO_PERFIL.REVISAO" />
</ElementType>

<ElementType name = "REVISAO_PERFIL.EMP_NUM" xa:fkey = "EMPLOYEE.NUM" content =
"textOnly" dt:type = "i4"/>
<ElementType name = "REVISAO_PERFIL.DATA_REVISAO" content="textOnly" dt:type = "date"/>
<ElementType name = "REVISAO_PERFIL.REVISAO" content = "textOnly" dt:type = "string"/>

<ElementType name = "MUDANCA_SALARIAL" content = "eltOnly" order = "seq">
   <AttributeType name = "MUDANCA_SALARIAL.EMP_NUM_idref" dt:type = "IDREF" required =
"yes"/>
   <attribute type = "MUDANCA_SALARIAL.EMP_NUM_idref"/>
   <element type = "MUDANCA_SALARIAL.EMP_NUM" />
   <element type = "MUDANCA_SALARIAL.DATA_REVISAO" minOccurs = "0" maxOccurs = "1" />
   <element type = "MUDANCA_SALARIAL.DATA_MUDANCA" />
   <element type = " MUDANCA_SALARIAL.SALARIO" />
</ElementType>

<ElementType name = "MUDANCA_SALARIAL.EMP_NUM" xa:fkey = "EMPLOYEE.NUM" content =
"textOnly" dt:type = "i4"/>
<ElementType name="MUDANCA_SALARIAL.DATA_REVISAO" content="textOnly" dt:type = "date"/>
<ElementType name="MUDANCA_SALARIAL.DATA_MUDANCA" content="textOnly" dt:type = "date"/>
<ElementType name=" MUDANCA_SALARIAL.SALARIO" content = "textOnly" dt:type = "i2"/>

</Schema>
```

4.5 – Concluindo o Exemplo

Adotando algumas convenções simples, schemas XML podem modelar com sucesso diversas fontes de informação, como os provenientes de bancos de dados relacionais.

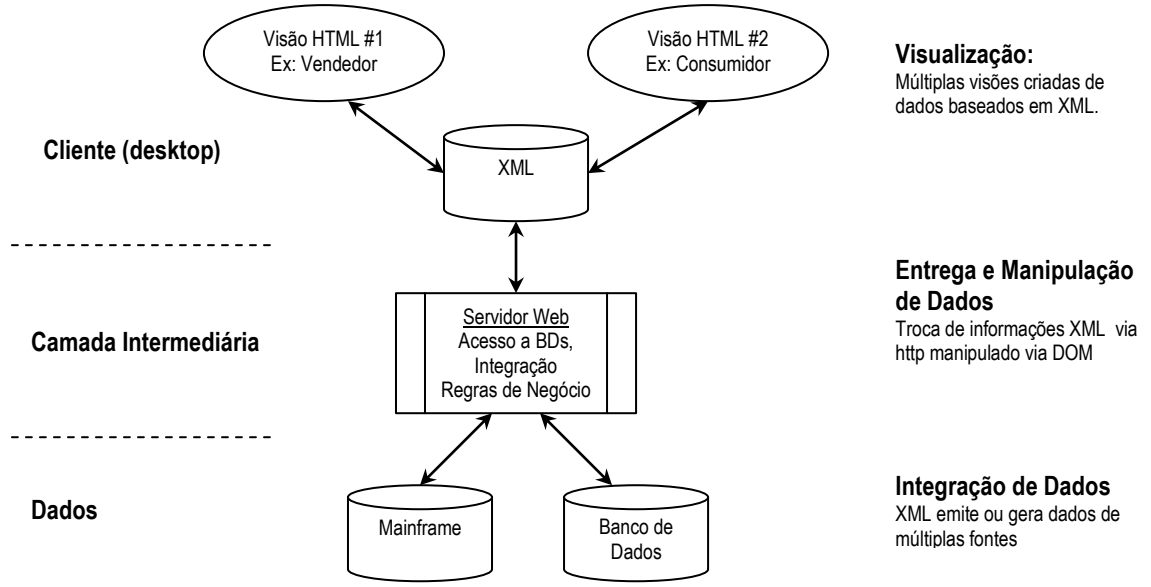


Figura 48 – Estrutura de Modelagem entre SGBD e XML.

Definido os tipos de dados, os relacionamentos e informação *id/idref*, pode-se armazenar metadados necessários a facilitar o processo de troca de informação para ambos os lados (XML e SGBDR).

O processo definido anteriormente é direto e bem automatizado e foi implementado por grande variedade de bancos de dados que suportam XML.

4.6 – Considerações sobre o capítulo

A facilidade apresentada na modelagem de estruturas completas de bancos de dados para XML e vice-versa mostra a viabilidade de se usar essa tecnologia como meio de comunicação e interação entre banco de dados heterogêneos.

Esse conceito (mapear estruturas e dados de banco de dados para um documento XML) pode gerar várias opções de utilização em diversas áreas, tais como comércio eletrônico, apoio a *extranets*, comércio *business to business* e *business to consumer*, entre muitos outros.

No capítulo que se segue, mostrar-se-á um exemplo de implementação de *middleware* cujo objetivo e mapear estruturas e dados de diferentes bancos de dados para documentos XML. O tipo de mapeamento desenvolvido foi o definido nesse capítulo, baseado tanto em elementos quanto em atributos.

Capítulo 5

UM ESTUDO DE CASO DE MIDDLEWARE SGBD/XML

Baseado nos conceitos de modelagem apresentados nos capítulos anteriores foi desenvolvido, para justificar a proposta dessa dissertação, um *middleware* que consiste na transformação de informações contidas em um SGBD relacional para XML e vice-versa. Esse *middleware* foi apelidado de SIMBA, ou Sistema de Interação e Manipulação de Banco de Dados.

Neste capítulo serão abordados todos os detalhes relacionados a esse software e o ambiente (plataforma) em que foi desenvolvido, apresentando resultados da análise do *middleware* juntamente com os problemas encontrados na solução, assim como pretensões futuras de continuação desse projeto, a fim de solucionar ou amenizar esses problemas.

5.1 – O Ambiente

O software foi desenvolvido em *Delphi 5.0 Enterprise Edition*, onde usou alguns componentes de terceiros para controlar o acesso a vários SGBDs. Esses componentes foram o Midas da Borland, e o Visibroker da VisiCorp. Ambos os componentes prove acesso a SGBDs via DCOM ou CORBA.

O DCOM ou *Distributed Commom Object Model* é desenvolvido pela Microsoft e conseqüentemente dá suporte a todos os SGBDs compatíveis com ela, seja o MS SQL Server, o Oracle, Sybase ou DB2.

Já o CORBA ou *Commom Object Request Broker Architeture* foi projeto por um conjunto de empresas não-Microsoft, como a IBM, Sun, Oracle e diversos fabricantes Unix. Com isso o CORBA dá suporte a praticamente todos os SGBDs do mundo Unix ou similar.

Pelo fato do SIMBA usar comunicação CORBA e DCOM simultaneamente, pode ser classificado como um modelo híbrido, o que acarreta um maior número de

SGBDs suportados, quer dizer, a maioria absoluta dos SGBDs atualmente ou são compatíveis com CORBA ou com o DCOM.

A lista de todos os SGBDs que o SIMBA suporta ainda não foi possível ser compilado, mas os testados foram:

- MS SQL Server,
- DB2,
- Informix,
- Interbase,
- Oracle e
- Sybase.

Outros tipos de fontes de dados também foram testados, como:

- MS Access,
- Paradox,
- FoxPro,
- Dbase,
- Excel e
- Arquivos Textos e
- Fontes de Dados ADO ou ODBC1.

Teoricamente, essas fontes de dados podem estar na maioria dos sistemas operacionais atuais, desde que dêem suporte a TCP/IP para comunicação. Entretanto, o SIMBA somente foi testado nas plataformas Windows e Linux.

[1] ADO (*Active Data Objects*) e ODBC (*Open Database Conectivity*) são produtos da Microsoft que possibilita a comunicação com diversos Banco de Dados.

Já o SIMBA roda somente em plataforma Windows (95, 98, NT e 2000), dado o fato de que o Delphi só dá suporte a esses ambientes de programação. Entretanto, nesse ano foi lançado o Kylix da Borland, que seria o Delphi para Linux. Futuramente pode-se ter uma versão do SIMBA para Linux.

5.2 – O Middleware

Como já foi apresentado, o SIMBA prove acesso a diversos SGBDs via CORBA e DCOM. Para tanto, ele foi desenvolvido no modelo multitier (ou 3 camadas), que seriam:

Camada 1: SGBDs e suas regras.

Camada 2: Servidor de Aplicações (CORBA ou DCOM).

Camada 3: Cliente Magro (Windows e futuramente Linux).

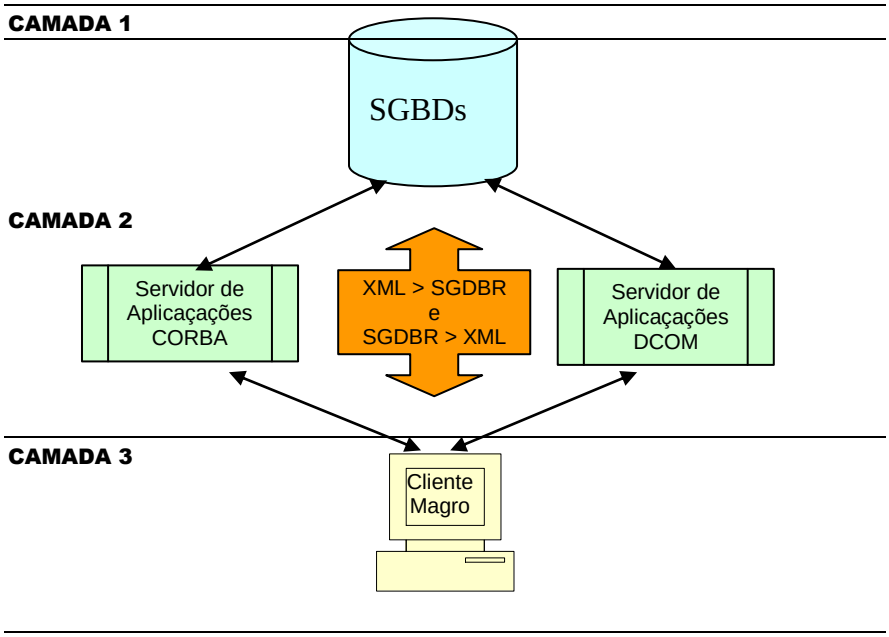


Figura 49 – As 3 camadas do SIMBA.

5.2.1 – Camada 1

Essa camada é onde o SGBD Relacional é ativado para prover informações sobre um determinado assunto. Essas informações podem ser somente tabelas simples,

ou podem ser um conjunto de informações mais detalhadas obtidas através de uma *View* ou *Stored Procedure*.

Nesse modelo, estuda-se somente banco de dados relacionais, dado a maior disponibilidade no mercado. Futuramente, pode-se analisar o comportamento do SIMBA com banco de dados orientados a objetos e hierárquico.

5.2.2 – Camada 2

É nessa camada que são desenvolvidos os servidores de aplicações, que podem ser provedores DCOM ou CORBA. Essa camada é inserida entre a aplicação em si e o SGBD, pois provê um maior controle na segurança e no acesso aos dados.

Nos servidores de aplicações podem-se definir regras de negócio onde diminuiria e muito a carga de processamento nos clientes. No caso do SIMBA, os servidores de aplicações provêem um encapsulamento das APIs de acessos CORBA e DCOM, facilitando e muito a interface da aplicação cliente (3ª camada) com os SGBDs (1ª camada).

Esses servidores podem rodar tanto em ambiente Windows quanto em alguns tipos de Unix (os compatíveis até então são o Linux, Solaris, HP-UX, AIX e SCO Unix). Os servidores de aplicações do SIMBA rodam em Windows, mas podem ser facilmente portados para outras plataformas dado que todas as regras de acesso e encapsulamento de APIs contidas nesses servidores foram desenvolvidos em IDL (*Interface Definition Language*), graças a um gerador IDL contido no Delphi.

Figura 50 – Servidores de Aplicações DCOM e CORBA.

5.2.3 – Camada 3

A 3ª camada consiste na aplicação cliente, que provê toda uma interface amigável para as informações a serem manipuladas, sejam elas obtidas por interface DCOM ou por interface CORBA.

Além disso, na aplicação cliente, foi implementado também o modelo de 2 camadas, onde a aplicação acessa diretamente o SGBD. Nesse caso a aplicação só suporta os SGBDs suportados pelo Delphi.

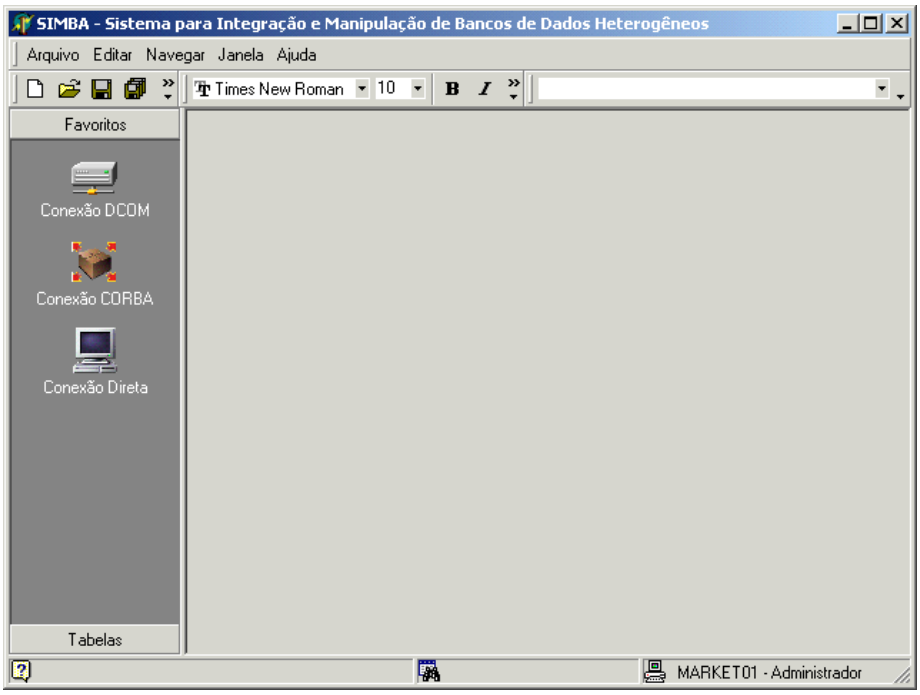


Figura 51 – Aplicação cliente - SIMBA

5.2.3.1 – Acessando Dados

Como se pode ver na figura anterior, o acesso aos dados se dá em 3 maneiras: via Conexão DCOM; Conexão CORBA e Conexão Direta. Basta clicar em um desses ícones que se abrirá uma janela com a conexão escolhida já ativa. Lembrando que os servidores DCOM e CORBA devem estar rodando em algum ponto da rede.

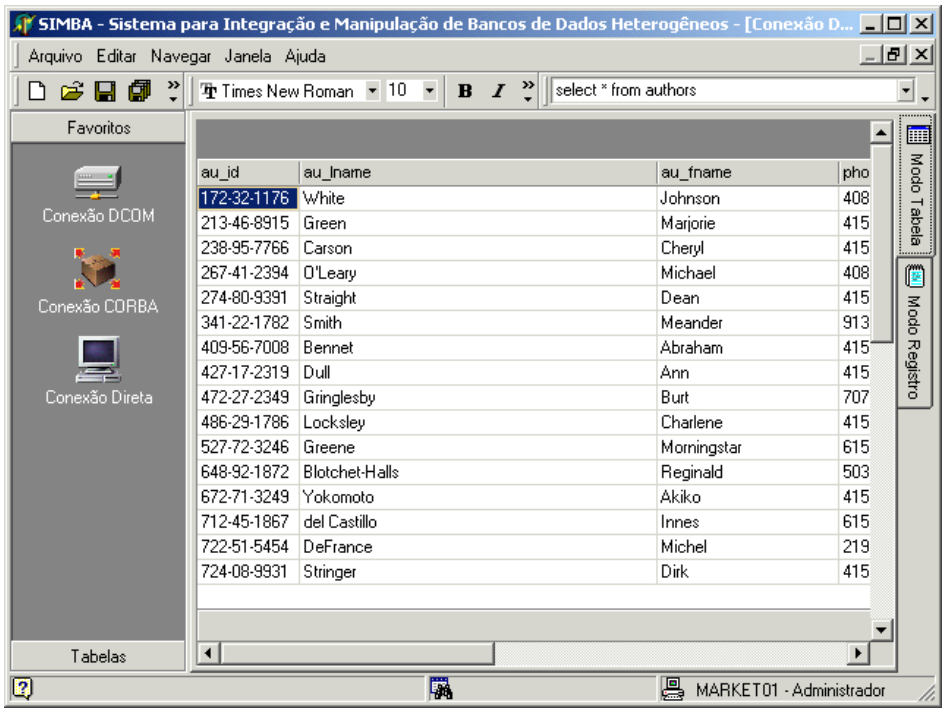
Depois de conectado, basta selecionar um dos comandos SQL no editor SQL localizado na barra de ferramentas, que o mesmo se encarregará de trazer as informações desejadas. No editor SQL, poderão ser inseridas consultas a *stored procedures* e *views*, desde que obedeçam aos padrões definidas pela ANSI SQL92.

Com a janela de dados aberta, podem ser feitos quaisquer procedimentos que são comuns com tabelas relacionais, como:

- Inclusão, alteração e exclusão,
- agrupamento,

- indexação e ordenação,
- inclusão e exclusão de colunas e linhas,
- localização de dados,
- filtagem de dados,
- totalização de informações entre muitos outros.

Tudo de uma maneira interativa e amigável.



au_id	au_lname	au_fname	pho
172-32-1176	White	Johnson	408
213-46-8915	Green	Marjorie	415
238-95-7766	Carson	Cheryl	415
267-41-2394	O'Leary	Michael	408
274-80-9391	Straight	Dean	415
341-22-1782	Smith	Meander	913
409-56-7008	Bennet	Abraham	415
427-17-2319	Dull	Ann	415
472-27-2349	Gringlesby	Burt	707
486-29-1786	Locksley	Charlene	415
527-72-3246	Greene	Morningstar	615
648-92-1872	Blotchet-Halls	Reginald	503
672-71-3249	Yokomoto	Akiko	415
712-45-1867	del Castillo	Innes	615
722-51-5454	DeFrance	Michel	219
724-08-9931	Stringer	Dirk	415

Figura 52 – Acessando dados via SIMBA.

5.2.3.2 – Transformandos dados em XML e em outros formatos

Depois dos dados recuperados e trabalhados, pode-se proceder com a conversão desses dados para documentos XML. Para tanto basta selecionar o Menu `Arquivo - Salvar Como`, que se abrirá uma caixa de diálogo com várias opções de formato, sendo que uma delas é a XML.

Algumas características foram adicionadas e que podem ser muito úteis, como salvar essas informações em mais de um tipo de documento. Esses tipos são:

- Arquivo Texto,

- Arquivo Binário (criptografado ou não)
- Planilha Excel,
- Página HTML e
- Arquivo PDF.

Alguns outros formatos poderão ser incluídos de acordo com as necessidades apresentadas.

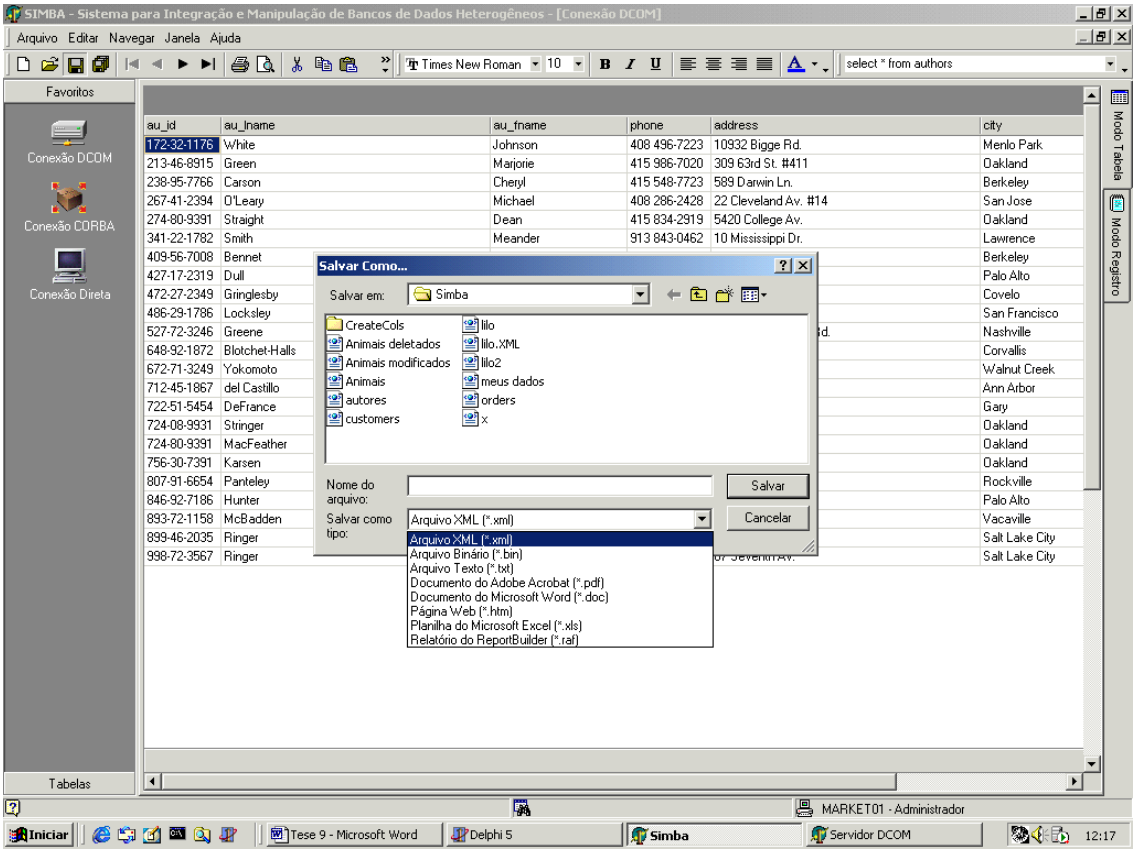


Figura 53 – Salvando dados em XML e outros formatos

5.2.3.3 – Recuperando dados de documentos XML e de outros formatos

Assim como se pode salvar quaisquer dados de SGBDs para os tipos de documentos definidos anteriormente, também pode-se recuperá-los, permitindo assim trabalhar com eles como se fossem tabelas, provendo as mesmas características que uma tabela relacional provê, como inclusão, alteração, exclusão, indexação, filtragem, etc.

Para tanto basta selecionar o Menu `Arquivo - Abrir`, que se abrirá uma caixa de diálogo com várias opções de formato, sendo que uma delas é a XML, além, de outros formatos já definidos anteriormente.

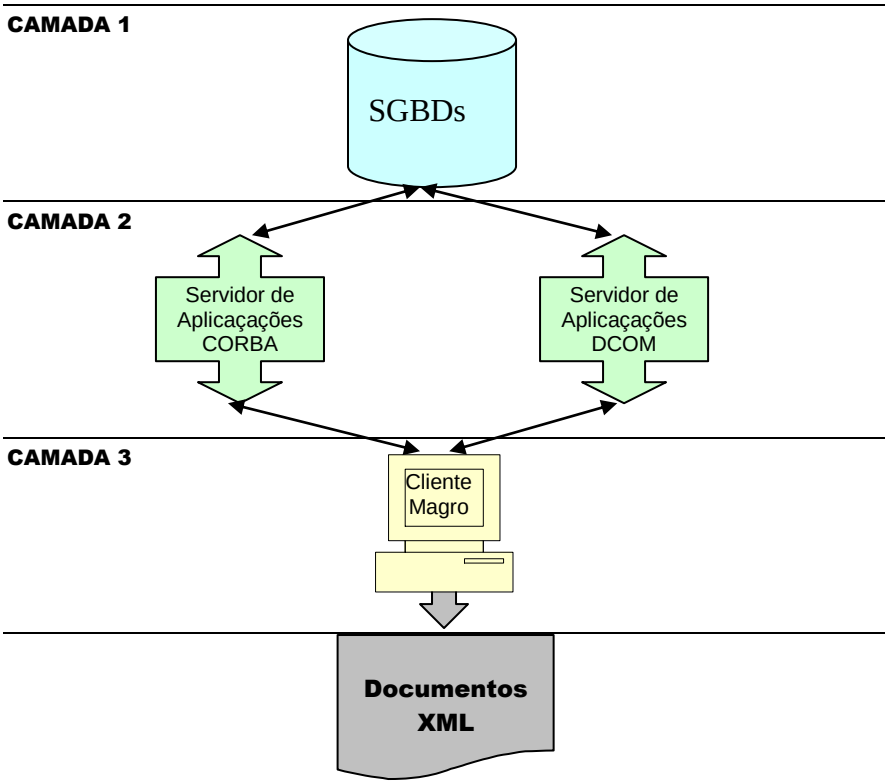


Figura 54 – Interface entre o SIMBA e XML

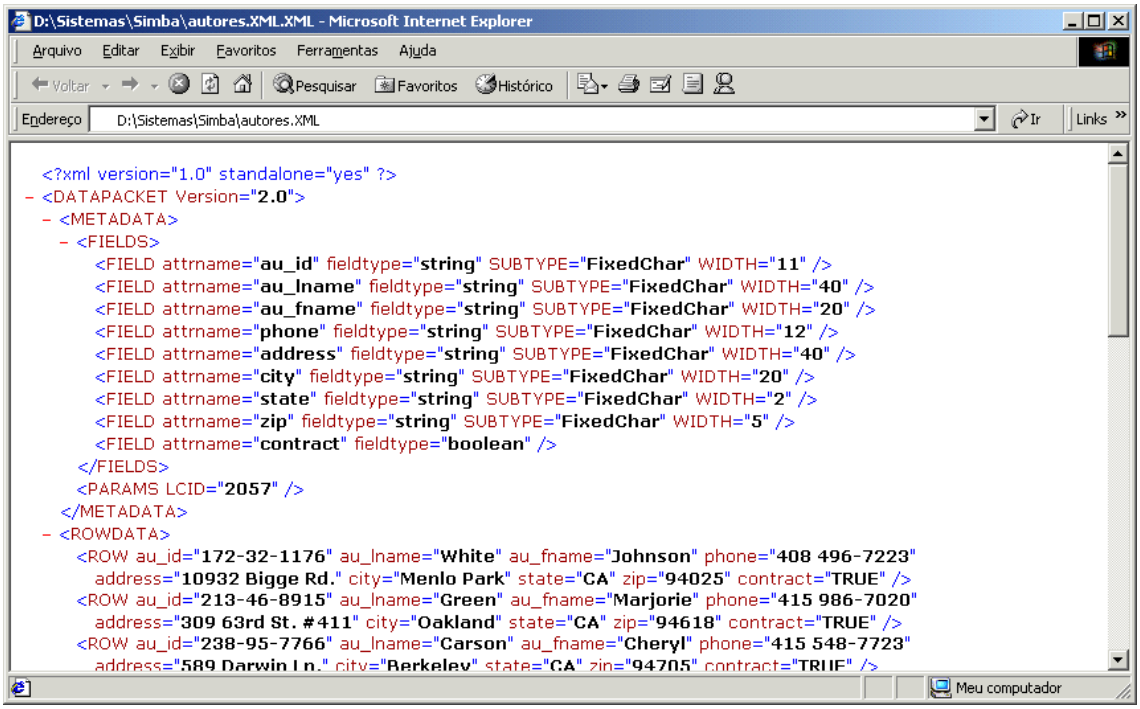


Figura 55 – Documento XML gerado via SIMBA

5.3 – Recursos não Implementados

Dois dos recursos mais importantes que ainda não foram desenvolvidos nesse *middleware* é a modelagem de banco de dados orientados a objetos e hierárquico (o SIMBA só trabalha atualmente com modelos relacionais e objeto-relacionais) e uma interface de atualização de documentos XML para os SGBDs.

5.3.1 – Modelagem de banco de dados orientados a objetos e hierárquico

O fato de não haver a possibilidade de se modelar dados provenientes de SGBD orientados a objetos e hierárquicos é o não suporte do Delphi para esses SGBDs. Entretanto, foi explicado com detalhes como se modela esses tipos de dados para XML no capítulo anterior, podendo ser base para futuras implementações.

5.3.2 – Atualização de XML para SGBD

A migração de documentos XML recuperados (ou quaisquer outros tipos providos) para SGBD é um recurso muito importante que ainda não foi implementado (dado a falta de tempo e de literatura sobre o assunto), é que é a base da integração de SGBDs via XML, pois é justamente o caminho de volta.

Essa implementação é um pouco complicada, por poder acarretar problemas com a integridade dos dados do SGBD, que podem surgir quando há uma inserção, atualização ou exclusão de dados.

5.3.2.1 - Inclusão

Acontece quando é necessário inserir um documento novo. Erros podem acontecer quando esse documento já estiver inserido no banco de dados.

Existem várias maneiras de se contornar essa situação, mas a mais utilizada é eleger o “melhor” documento para ser o definitivo. Lembrando que não existe uma regra bem definida de como ocorre essa “eleição”, mas geralmente é garantido para o documento que foi inserido primeiro. Sendo que é notificado para a fonte que gerou o segundo documento, uma falha na inclusão do mesmo, apresentando o motivo dessa falha.

5.3.2.2 - Alteração

Acontece quando é necessário alterar um documento não existente. Erros podem acontecer quando o documento a ser alterado não estiver no banco de dados, ou se não for definido uma chave de acesso para atualização.

Também existem várias maneiras de se contornar tal situação, mas também não se pode garantir qual delas é a mais apropriada, isso vai muito dos negócios definidos de um SGBD para outro.

Uma maneira de resolver esse problema é que se não encontrar o documento na qual se quer alterar, basta inseri-lo do banco de dados, outra alternativa seria gerar um erro para a fonte que originou tal documento.

Quando o motivo é o não fornecimento de uma chave de acesso (chave primária) para poder encontrar o documento em questão, então se pode localizar o documento fazendo uma comparação valor a valor, ou simplesmente gerando um erro. No primeiro caso, o mesmo só é eficaz se for encontrado apenas um documento com os valores pretendidos, mesmo assim ainda está sujeito a problemas, já que não haverá uma certeza se aquele é ou não é o documento a ser alterado.

5.3.2.3 - Exclusão

Acontece quando é necessário excluir um documento não existente. Erros podem acontecer quando o documento a ser excluído não estiver no banco de dados, ou se não for definido uma chave de acesso para exclusão.

As maneiras de se contornar tais situações são similares ao da alteração, sendo a alternativa mais pertinente é gerar um erro (exceção) para a fonte desse documento.

5.4 – Considerações sobre o modelo

Apesar de faltar uma pequena parte da implementação do modelo, pode-se afirmar que é extremamente viável a transformação de dados provindos de SGBDs para XML e vice versa, dado a análise verificada no modelo implementado.

A grande vantagem dessa modelagem, é que o documento gerado é um padrão mundial, e que pode ser analisado e obtido a partir de qualquer SGBD seja ele

relacional, orientado a objetos, hierárquico e até mesmo banco de dados legado, pois há uma tendência para que todas as plataformas suportem XML.

Além disso, é um documento que pode utilizar a Internet via meio de comunicação, que é uma alternativa muito barata, facilmente implementada e com resultados extremamente satisfatórios, o que pode ser verificado em grandes portais *Business to Business*, *Business to Consumer*, e em alguns sites da web.

Outros softwares e tecnologias disponíveis para o uso de XML com banco de dados estão apresentados no Apêndice A dessa dissertação.

Capítulo 6

CONCLUSÃO

O principal objetivo dessa dissertação foi a apresentação de um novo modelo para comunicação e interação de banco de dados heterogêneos Para tal, usamos a tecnologia XML que fornece meios de integração de dados e informações provindas desses SGBDs.

Para atingir este objetivo foi realizado um amplo estudo sobre as tecnologias nas quais se baseiam esse modelo que são a interação entre SGBDs (apresentado no Capítulo 1) e a XML (apresentado no Capítulo 2). Também foi estudado as relações que podem existir entre os dois (apresentado no Capítulo 3), mostrando como se pode aplicar uma tecnologia (XML) em favor de um conceito (modelagem e integração). Depois, no Capítulo 4, foi definido um modelo de mapeamento de informações de um banco de dados para XML e vice-versa, e por último, no capítulo 5, foi descrito um *middleware* que foi desenvolvido a partir dos conceitos de modelagem apresentados nessa dissertação, que teve como objetivo, comprovar a facilidade de implementação do modelo e garantir a comunicação e interação entre banco de dados heterogêneos.

6.1 – Fundamentação

Os SGBDs estão bem modernos no que se referem a manipulação dos dados, provendo segurança, transações, integridade, acesso de múltiplos usuários, linguagens de consultas robustas, etc., entretanto quase todos eles são montados em cima de uma abordagem proprietária.

Até mesmo a linguagem de consulta que deveria ser padronizada (via ANSI SQL 92 e mais recentemente SQL 3) são diferentes de SGBD para SGBD, onde cada um implementa funções específicas em sua plataforma a fim de prover mais eficiência na manipulação de seus dados.

Sabe-se também que com a globalização hoje e as crescentes necessidades dos usuários, cada vez mais complexas e elaboradas, praticamente “obrigam” que esses dados sejam disponibilizados ao longo de toda uma organização de negócio (que tende a

ter alcance mais amplo graças a Internet), e em casos mais complexos, que esses dados estejam integrados com outras fontes de dados com o objetivo de fornecer mais informações para apoio ao negócio e a tomada de decisões.

Dado essas necessidades, surgiu o conceito de *datawarehouse* e *datamining*, e conseqüentemente, vários fabricantes fornecendo tal tecnologia. Mas como se viu, implementações desse tipo de conceito requer muito tempo e esforço, o que acarreta custos altos e cujos resultados, muitas vezes não são tão satisfatórios (já que a solução tem que ser muito bem elaborada antes de aplicada).

O que essa dissertação tenta argumentar é que não é necessário buscar soluções tão caras e complexas como *datawarehouse* para prover um meio de comunicação e integração com diferentes bancos de dados, mesmo com informações heterogêneas, já que se pode usar o XML, que é poderoso o suficiente para prover tais recursos.

6.2 – Pontos Fortes e Pontos Fracos

Foi constatado ao longo do estudo que existem problemas na utilização da XML como meio de integração, isso porque esse modelo não se baseia em dados em tempo real, o que dificultaria e muito o controle ao longo de transações que envolvem respostas de dados on-line e grande concorrência.

Entretanto, a adoção desse modelo facilmente se molda a maioria das situações que normalmente são encontrados em modelos reais.

Isso porque a XML é capaz de modelar praticamente qualquer tipo de informação, seja ela contida em um documento ou na forma de dados puro, advinda tanto de SGBDs relacionais, quanto orientado a objetos ou hierárquico, e isso tudo baseado em um modelo altamente padronizado e aberto, que é essencial para a integração entre diferentes plataformas.

E o mais importante, a XML nasceu para a Web, isto é, o seu uso implica em todas as vantagens que a internet provê como independência de plataforma, velocidade, abrangência, simplicidade e muito mais.

É por isso que essa dissertação se ateve a esses conceitos, pois uma das maneiras mais eficientes e eficazes de se criar um meio de comunicação e interação entre SGBDs, é a XML, e o mesmo já está se tornando padrão de mercado e amplamente suportado pelos sistemas de banco de dados (é só ver as novas aplicações de e-commerce, B2B, B2C, etc.).

Apesar da tecnologia XML ser nova, ela já evoluiu bastante e continua evoluindo graças aos esforços mundiais que tendem a criar padrões abertos que estejam disponíveis a todas as organizações e pessoas.

6.3 – Perspectivas Futuras

Como resultado desse trabalho chegou-se a criação de um modelo que mapeia informações de um banco de dados relacional para XML e vice-versa (apresentado no Capítulo 5). Foi utilizado o Delphi como plataforma de desenvolvimento, DCOM e CORBA como meio de comunicação entre a aplicação e os BDs, e vários tipos de SGBDs para garantir um ambiente de heterogeneidade.

A dificuldade em se criar esse modelo foi a falta de especificações e literatura disponíveis, mesmo já existindo várias empresas que fornecem tecnologias baseadas em XML e SGBD, todas elas escondem em suas documentações especificações técnicas acerca do modelo. E esse foi um dos principais motivos de se implementar uma nova aplicação *middleware* ao invés de analisar uma já existente, além do fato de que a maioria dos fabricantes desta tecnologia cobram pelos seus softwares.

Analisando os resultados do projeto fruto dessa dissertação, chegou-se a conclusão de que é possível a comunicação entre banco de dados heterogêneos via XML e conseqüentemente sua interação, e o mais importante, esse modelo se apresentou como sendo bem viável e eficiente, dados os padrões abertos que se baseia.

No projeto de desenvolvimento, alguns pontos não foram implementados, como o controle de transações e conflitos de inserção, atualização e exclusão, e deixando isso para trabalhos futuros.

Também em trabalhos futuros, pretende-se coletar informações da integração dos sistemas e informações baseado na administração pública estadual e municipal,

gerando modelos de referência e métodos de implantação. Nesse processo, verificar inclusive a adoção do SIMBA e seu comportamento em um ambiente real, com várias plataformas heterogêneas que não se comunicam diretamente, onde métodos de conexão CORBA e DCOM serão de grande valia.

Além disso, pode-se avaliar o uso de outras tecnologias que provêem praticamente o mesmo resultado que a XML na comunicação e interação entre banco de dados heterogêneos. Nessa avaliação se poderia analisar com mais fundamentação científica os prós e contras da adoção de cada modelo alternativo, em comparação a XML (apesar do mercado já adotar a XML como padrão).

Apêndice A

PRODUTOS QUE SUPORTAM A TECNOLOGIA XML/SGBD

O número de produtos que usam XML com bancos de dados está crescendo com velocidade surpreendente (produtos novos parecem entrar no mercado semanalmente).

Pode-se dividir esses produtos nas seguintes categorias:

- *Middleware*: Software que é chamado de sua aplicação para transferir dados entre documentos XML e banco de dados.
- *Banco de Dados Compatíveis com XML*: São bancos de dados com extensões para transferir dados entre documentos XML e eles mesmos.
- *Banco de Dados XML (nativo)*: São bancos de dados que armazenam XML em sua forma nativa.
- *Servidores XML*: São plataformas que provêem dados (na forma de documentos XML) de e para aplicações distribuídas, como aplicações de comércio eletrônico e business to business.
- *Servidores de Aplicação XML*: São aplicações web que provêem XML (usualmente construídas de páginas web dinâmicas) para navegadores.
- *Sistemas de Administração de Conteúdo*: São sistemas que gerenciam fragmentos de documentos legíveis e inclui suporte para edição, controle de versão, e construção de novos documentos a partir de fragmentos já existentes.

Em geral, é necessário escrever código para integrar *Middleware*, Bancos de dados Compatíveis com XML, Bancos de dados XML Nativos e Servidores de XML com suas aplicações. Servidores de Aplicação XML exigem que se façam alguns ajustes, e Sistemas de Administração de Conteúdo precisam ser configurados, o que pode ser uma tarefa não muito trivial.

No entanto, será abordado nesse apêndice somente produtos do tipo *Middleware*, na qual o SIMBA se inclui (cuja implementação foi baseada nos conceitos abordados nessa dissertação). Dificilmente se poderia analisar cada uma dessas ferramentas, pois são em sua maioria pagas e, quando disponibilizados para avaliação, escondem aspectos técnicos de sua implementação, que é o que importa.

A.1 – Middleware

Middleware é software usado por aplicações centrado a dados para transferir dados entre documentos XML e bancos de dados. É escrito em uma variedade de linguagens, mas quase todos usam ODBC, JDBC, CORBA ou OLE DB. Embora alguns destes podem enviar dados pela Internet, a maioria precisa ser usada com um servidor Web, caso precisar de acesso de dados remoto.

A.1.1 – ALLORA

Fabricante: HiT Software

URL: http://www.hitsw.com/dsheets/alloramidware.htm

Licensa: Comercial

Banco de Dados Suportados: Relacional via OLE DB, ODBC e JDBC

Direções: Banco de Dados para XML e XML para Banco de Dados

Allora é um middleware que aplicações podem chamar para recuperar dados de uma tabela, conjunto de resultados, ou consulta de catálogo como uma árvore DOM Nível 2 ou como eventos SAX 2. O documento XML é modelado como uma única tabela, com colunas mapeadas como elementos filhos.

Allora está disponível nas versões Java e Windows. Ambos têm essencialmente as mesmas características. Entretanto a versão de Java tem algumas opções a mais sobre como a árvore de DOM é construída (nomes de tipo de elemento para usar, coluna para designar como um atributo ID, inclusão de um DTD, etc.). Também pode trilhar mudanças para a árvore DOM e deste de volta para o banco de dados. A versão de Windows vem com um visualizador gráfico de banco de dados e uma ferramenta para localizar e trilhar chamadas de API Allora.

A.1.2 – ASP2XML

Fabricante: Stonebroom

URL: http://www.stonebroom.com/asp2xml.htm

Licensa: Comercial

Banco de Dados Suportados: Relacional via OLE DB e ODBC

Direções: Banco de Dados para XML e XML para Banco de Dados

Um objeto COM para transferir dados entre documentos XML e fontes de dados ODBC ou OLE DB. O produto é dirigido a modelo e o documento de XML é modelado como uma única tabela. Ao transferir dados do banco de dados para XML, o usuário especifica uma única declaração SELECT e a saída contém *tags* específicas ASP2XML que presumivelmente podem ser descartadas. Ao transferir dados de XML para o banco de dados, o documento XML tem que conter *tags* específicas ASP2XML que são requeridas para processamento. O objeto pode ser usado em páginas ASP (Microsoft Active Server Pages) ou sozinhos.

A.1.3 – BEANSTALK

Fabricante: Transparency

URL: http://www.transparency.com/

Licensa: Comercial

Banco de Dados Suportados: Relacional via ODBC

Direções: Banco de Dados para XML e XML para Banco de Dados

Uma máquina objeto-relacional que se situa entre a aplicação e o banco de dados. O produto estende a sintaxe do SELECT do SQL3, permitindo *subselects*. Isto possibilita ao produto construir uma árvore de conjuntos de resultado que são profundamente aninhados.

Para recuperar resultados como XML, o usuário deve incluir uma função XML() na declaração SELECT. O usuário pode especificar se o SELECT devolve resultados como elementos ou atributos, assim como definir também os nomes desses elementos e atributos. Uma sintaxe semelhante é usada para inserir dados de um documento de XML para o banco de dados.

A.1.4 – DATABASEDOM

Fabricante: IBM

URL: http://www.alphaworks.ibm.com/tech/databasedom

Licensa: Somente cópia de demonstração

Banco de Dados Suportados: DB2 e Microsoft Access

Direções: Banco de Dados para XML e XML para Banco de Dados

É um *JavaBean* que transfere dados entre uma árvore DOM e uma única tabela em um banco de dados. O produto é dirigido a *template*, com seções separadas no modelo para a informação de acesso de banco de dados e o layout de dados. Ao transferir dados do banco de dados para a árvore DOM, o usuário especifica uma declaração SELECT ou um nome de tabela e a cláusula WHERE, quando transferir dados da árvore DOM para o banco de dados, o usuário pode especificar só um único nome de tabela.

A.1.5 – ADO

Fabricante: Microsoft

URL: http://msdn.microsoft.com/xml/articles/xmlintegrationinado.asp

Licensa: Comercial

Banco de Dados Suportados: Relacional via OLE DB e ODBC

Direções: Banco de Dados para XML e XML para Banco de Dados

ADO pode persistir um objeto *Recordset* como um documento de XML. Também pode abrir um documento de XML como um objeto *Recordset*. Isto provê um modo para transferir dados entre XML e um banco de dados, usando *Recordsets* como objetos intermediários.

O documento de XML é divido em duas partes. A primeira parte mapeia a estrutura do Recordset. A segunda parte contém os dados atuais em formato XML. O mapeamento e dirigido a modelo, com os dados modelados como uma árvore de objetos específicos de dados. Uma coisa agradável sobre o uso de ADO com XML é que uma árvore de elementos aninhados é aberta como uma árvore de *Recordsets* aninhados e vice-versa.

Se o *Recordset* contiver atualizações, exclusões ou inclusões pendentes, estes especificamente são sinalizados no documento XML com etiquetas específicas do ADO. No caso de atualizações, são incluídos tanto os dados originais quanto os novos.

Apêndice B

ESTADO DA ARTE EM COLABORAÇÃO E COMUNICAÇÃO VIA XML

O grau de maturidade da segurança de protocolos baseados na Internet, combinada com apoio onipresente para estes protocolos pelas redes, hardwares, e softwares, está permitindo desenvolver novos mecanismos (tecnologia, softwares, etc.) para facilitar interações eficientes e automatizadas de informações intra e inter-organizacionais, tais como interações entre os negócios internos da organização, entre aplicações de administração de conhecimento e produtividade, entre aplicações usadas pelos clientes e parceiros, e entre serviços fornecidos pelos seus provedores corporativos e comerciais.

Essa segurança é extremamente necessária, pois diferentemente do que havia antes (integração de informações interno a organização), uma dimensão nova de desafios nas áreas de segurança e confiança deve ser tratada para comunicar com outras organizações.

Atento a essas necessidades, algumas plataformas de integração de informações já estão presente no mercado, funcionando de maneira semelhante ao que foi apresentado nessa dissertação (mapeando informações internas da organização armazenadas em banco de dados para documentos XML e enviando-os a seus parceiros organizacionais via internet). A principal delas, pelo menos a mais presente e comentada no mercado, é o BizTalk Server da Microsoft.

A seguir têm-se algumas características dessa solução, entretanto não foi possível analisar os aspectos técnicos do mapeamento dessas informações, pois as mesma não estão disponíveis nos manuais técnicos. O mesmo ocorre com praticamente todos os softwares desse tipo, o que foi mais um incentivo para escrever essa dissertação, que seria desvendar a tecnologia por traz do mapeamento de informações contidas em banco de dados para o XML.

B.1 Microsoft BizTalk Server

B.1.1 – Introdução

O XML e linguagens baseadas em schemas XML proporcionam um conjunto forte de tecnologias baratas e de fácil uso e aplicação. Estas linguagens permitem uma pessoa a descrever a troca estruturada de informações entre aplicações colaborativas ou parceiros de negócio em uma plataforma neutra.

Como resultado, iniciativas de indústria começaram a adotar XML e linguagens de schema baseados em XML para especificar os vocabulários e seus modelos de conteúdo. Este schemas são amplamente publicados e implementados para facilitar comunicação entre aplicações e negócios. O alto suporte ao XML também resultou em provedores de solução independentes que habilitam a troca de informação baseada em XML com outros provedores ou aplicações.

Para tanto o BizTalk Sever tem como principais objetivos prover via XML:

- Um idioma universal suficientemente flexível e rico para especificar, empacotar, publicar, e trocar informações estruturadas e não estruturadas através de suas aplicações e limites de negócio.

- Um idioma universal flexível e rico para especificar e executar réguas de transformação para converter informações de um formato para o outro.

- Uma plataforma neutra, protocolos de comunicação a nível de aplicação que habilitam interações automatizadas por aplicação ou limites de negócio.

- Mecanismos baseados em uma plataforma neutra de envio de mensagens seguras para integridade, privacidade e não-repúdio.

B.1.2 – Especificações

Documentos BizTalk segue um conjunto de regras para estrutura e conteúdo para prover alta funcionalidade e semântica previsível. Estas regras obedecem a seguinte semântica:

- Estrutura global de Documentos BizTalk.
- Cabeçalhos BizTalk por encaminhamento de documentos, propriedades, catálogos, e administração de processo.
- Estruturação e manuseamento de documentos BizTalk que requerem entrega segura.

Quando se implementa soluções que usam a plataforma BizTalk, transporte específico, codificação, e mecanismos de segurança devem ser usados para assegurar a entrega de mensagens. Esta especificação descreve os seguintes mecanismos e aspectos decodificação e transporte de mensagens do BizTalk:

- Mecanismos de transporte para protocolos de transferência da Internet
- Entrega de mensagens de modo seguro
- Empacotamento e transferência baseado em MIME
- Assinaturas e encriptação baseado em S/MIME
- Compatiblidade com SOAP (*Simple Object Access Protocol*), e Schemas XML (Estruturas e Tipos de Dados).

B.1.3 – O Uso de Tipos de Dados de Schemas XML

Esta especificação usa o atributo `xsi:type` como também vários tipos de dados próprios das especificações de Schema XML. Porém, esta especificação não define um método específico para criar schemas XML.

O atributo xsi:type permite que um elemento afirme explicitamente seu tipo em um documento XML. Isto pode ser usado para validar a estrutura do elemento. Alguns

tipos importantes são apresentados a seguir, sendo que os mesmos servem para representar praticamente qualquer tipo de dado.

O tipo de dados `timeInstant` representa um momento específico de tempo baseado no padrão ISO 8601. O tipo de dados de `uriReference` representa uma referência URI (*Uniform Resource Identifiers*) que pode ser absoluta ou relativa, e pode ter um identificador de fragmento opcional. Um `complexType` é um elemento cujo conteúdo não é um tipo simples, como uma string ou um número decimal; o elemento pode conter subelementos e/ou atributos. Os tipos de dados do schema XML são referenciados no texto sem o uso do prefiso `xsd:`.

Em comparação ao modelo desenvolvido nessa dissertação, existem várias semelhanças, já que os dois foram baseados nas mesmas especificações padrões, o SOAP Version 1.1 [SOAP], Schema XML Parte 1: Estruturas [XSD1], Schemas XML Parte 2: Tipo de Dados [XSD2], ISO 8601: Representações de datas e horas [ISO8601] , e Uniform Resource Identifiers (URI): Sintaxe genérica [URI].

B.1.4 – Conceitos

As principais partes do BizTalk são:

- **BizTalk Framework Compliant (BFC) Server**. Um Servidor BFC é representado por um conjunto de serviços que provêem a funcionalidade de processamento de mensagens definido nas especificações da base do BizTalk.

- **Application**. Uma Aplicação é o sistema onde as informações ou lógica são armazenadas e são executadas. Uma aplicação também inclui qualquer adaptador adicional que pode ser requerido para emitir ou receber Documentos Empresariais e comunicar com um servidor BFC.

- **Business Document**. É um documento XML bem-formado contendo dados empresariais. Estes dados podem representar uma ordem de compra, fatura, previsão de vendas, ou qualquer outra informação empresarial. Um ou mais documentos formam um documento BizTalk.

O BizTalk não define o conteúdo ou estrutura (schema) de Documentos de Negócio individuais. Os detalhes do conteúdo de Documento Empresarial e estrutura, ou Schema, é definido em acordo com as entidades empresariais envolvidas.

O modelo de implementação lógico para o BizTalk é composto de três camadas. Como a especificação BizTalk somente especifica o formato de transmissão de mensagens via um protocolo seguro, podem ser usadas camadas lógicas alternativas, provendo mais funcionalidades, sem afetar complacência com a especificação do BizTalk. Estas camadas lógicas incluem a aplicação (e adaptadores apropriados), o Servidor BFC, e transporte. A aplicação é a última fonte e destino do conteúdo de Mensagens de BizTalk, e comunica com outras aplicações enviando Documentos por Servidores BFC. Múltiplos servidores BFC comunicam um com outro em cima de uma variedade de protocolos, como HTTP, SMTP, e Microsoft Mensage Queue (MSMQ). O BizTalk não define o que estes protocolos de transporte são, e é independente de seus detalhes de implementação.

A aplicação é responsável para gerar os Documentos e qualquer anexo ser transmitido a seu(s) parceiro(s) e os submetendo ao Servidor BFC. A responsabilidade por embrulhar os Documentos em um Documento de BizTalk pode ser compartilhada tanto com a aplicação ou com o servidor BFC, dependendo da implementação do servidor BFC. O servidor processa o documento e qualquer anexo e constrói uma Mensagem BizTalk apropriado para o protocolo de transporte. O Servidor BFC usa informação contida no BizTags para determinar o endereço de destino. O servidor dá a mensagem então à camada de transporte para transmissão. As interfaces entre a aplicação, o Servidor BFC, e a camada de transporte, são implementações específicas.

O SIMBA, desenvolvido a partir dessa dissertação, também opera de modo semelhante, onde os servidores de comunicação são baseados em padrões DCOM e CORBA, e o mapeamento de informações empresariais para documentos XML bem formados, foi construído a partir das especificações do apresentadas no capítulo 4 dessa dissertação.

B.1.5 – Estrutura de um Documento BizTalk

Um Documento BizTalk consiste no padrão SOAP 1.1 que contém o seguinte:

- Um Documento específico (neste caso uma ordem de compra de livro), com seu próprio namespace XML definido, definido no corpo da mensagem SOAP.

- Específicas entradas de cabeçalho SOAP (<endpoints> e <properties>), construído usando BizTags, com schema e semântica definidas nesta especificação

```
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Header>
       <eps:endpoints SOAP-ENV:mustUnderstand="1"
                    xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
                    xmlns:agr="http://www.trading-agreements.org/types/">
           <eps:to>
               <eps:address xsi:type="agr:department">Book Orders</eps:address>
           </eps:to>
           <eps:from>
               <eps:address xsi:type="agr:organization">Book Lovers</eps:address>
           </eps:from>
       </eps:endpoints>
       <prop:properties SOAP-ENV:mustUnderstand="1"
                    xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
           <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</prop:identity>
           <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
           <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
           <prop:topic>http://electrocommerce.org/purchase_order/</prop:topic>
       </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <po:PurchaseOrder xmlns:po="http://electrocommerce.org/purchase_order/">
            <po:Title>Essential BizTalk</po:Title>
        </po:PurchaseOrder>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 56 – Exemplo de um documento BizTalk

B.1.6 – Corpo de um Documento BizTalk

O elemento <body> da mensagem SOAP que constitui um Documento BizTalk contém todos os Documentos a serem transportados. Em geral, um Documento BizTalk pode levar um conjunto de Documentos Empresariais relacionados (por exemplo, uma ordem de compra, o nome de um vendedor e o endereço de entrega daquela compra).

Documentos relacionados compartilharam freqüentemente conteúdo. O SOAP tem um mecanismo direto para codificar dados marcados por múltiplas referencias, pois usa os atributos ID do XML e URIs relativos. Considere um simples exemplo de ordem de compra, onde tempos informações da compra e da remessa de um livro. O elemento <body> do documento BizTalk a seguir mostra como o mesmo poderia ser expresso usando as regras de codificação SOAP.

```
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Header>
        <!-- headers omitted for brevity -->
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <po:PurchaseOrder xmlns:po="http://electrocommerce.org/purchase_order/">
          <po:item href="#theBook"/>
          <!-- and other purchasing information -->
      </po:PurchaseOrder>

      <ship:shippingInfo xmlns:ship="http://electrocommerce.org/shippingInfo/">
          <ship:content href="#theBook"/>
          <!-- and other shipping information -->
      </ship:shippingInfo>

      <book xmlns="http://electrocommerce.org/bookInfo/"  id="theBook"
                                                          SOAP-ENC:root="0">
          <Title>Essential BizTalk</Title>
          <!-- and other book information -->
      </book>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 57 – Exemplo de um documento BizTalk

Tanto na estrutura, quanto no corpo, existem similaridades encontradas com os documentos gerados pelo SIMBA, tanto que esses documentos podem ser transportados e manipulados pelo BizTalk, já que usam schemas XML padrões, definido pelo W3C.

Entretanto o SIMBA apenas gera documentos XML bem formados a partir de informações armazenadas em banco de dados relacionais, não se preocupando como se dará seu transporte e a segurança do mesmo. A união do SIMBA com o BizTalk seria de grande valia pois os serviços ainda não implementados pelo SIMBA poderiam ser usados sem problemas no BizTalk.

B.1.7 - Conclusão

Sabe-se que o BizTalk é uma ferramenta completa para conversão, criação e transporte de informações empresariais via documentos XML. Nessa dissertação, entretanto, foi estudado apenas um modelo de mapeamento (conversão) de dados armazenados em banco de dados relacionais para documentos XML, e a partir desse modelo foi criado um sistema – SIMBA – que o utilizasse e comprovasse sua eficácia.

Como o SIMBA é baseado em padrões abertos (definidos pelo W3C), todos os sistemas, soluções e plataformas que atendam as especificações padrões (o que inclui o BizTalk) poderão usá-lo como middleware alternativo, para manipulação de informações contidas diretamente a partir de um SGBD Relacional.

Apêndice C

PROCEDIMENTOS FORMAIS PARA O MAPEAMENTO ENTRE DADOS E XML

C.1 – Mapeando a Estrutura XML para a Estrutura do Banco de Dados

Para transferir dados entre um documento XML e um banco de dados, é necessário mapear a estrutura do documento com a estrutura do banco de dados e vice-versa. Tais mapeamentos entram em duas categorias gerais: *dirigido a template*[1] e *dirigido a modelo*.

C.1.1 – Mapeamento Dirigido a Templates

Em um *mapeamento dirigido a templates*, não há mapeamento pré-definido entre estrutura do documento e estrutura do banco de dados. Ao invés disso , embute-se comandos em um modelo (no caso um *template*) que é processado pelo middleware de transferência de dados.

Por exemplo, considere o seguinte modelo (não usado por qualquer produto real) nas quais as declarações *SELECT*[2] são embutidas dentro dos elementos `<SelectStmt>`:

```
<?xml version="1.0"?>
<VooInfo>
   <Intro>Os seguintes vôos tem poltronas disponíveis:</Intro>
   <SelectStmt>
      SELECT Airline, FltNumber, Depart, Arrive FROM Flights
   </SelectStmt>
   <Conclusao>
      Esperamos que estas satisfaçam seus anseios
   </Conclusao>
</VooInfo>
```

Figura 58 – Exemplo mapeamento dirigido a templates.

Quando processado pelo *middleware* de transferência de dados, cada declaração *SELECT* poderia ser substituída pelos seus resultados, formatada em XML:

```
<?xml version="1.0"?>
<VooInfo>
```

[1] É como se fosse um modelo já pré-definido
[2] SELECT é um comando SQL que é usado para fazer consultas em um banco de dados relacional.

```
      <Intro> Os seguintes vôos tem poltronas disponíveis:Intro>
      <Voos>
         <Linha>
            <EmpresaAerea>ACME</EmpresaAerea>
            <NumVoo>123</NumVoo>
            <Partida>Dec 12, 1998 13:43</Partida>
            <Chegada>Dec 13, 1998 01:21</Chegada>
         </Linha>
      </Voos>
      <Conclusao>
         Esperamos que estas satisfaçam seus anseios
      </Conclusao>
   </VooInfo>
```

Figura 59 – Exemplo mapeamento dirigido a templates processado pelo middleware.

Mapeamento dirigido a templates podem ser tremendamente flexíveis. Por exemplo, alguns produtos (*middleware*) lhe permitem colocar conjuntos de valores resultantes de um banco de dados em qualquer lugar no documento resultante[1], que é muito mais prático do que os próprios documentos terem que formatar seus resultados, como é mostrado acima.

Alguns fornecedores de *middleware* suportam declarações para *loops*[2] e *ifs*[3], inclusive fornecendo suporte a parametrização de declarações SELECT, assim como suporte para parâmetros HTTP[4].

Atualmente, *mapeamento dirigido a templates* só estão disponíveis para transferir dados de um banco de dados (relacional) para um documento de XML.

C.1.2 – Mapeamento Dirigido a Modelo

Em um *mapeamento dirigido a modelo*, um modelo de dados de algum tipo é definido na estrutura do documento XML, e isto é mapeado, implicitamente ou explicitamente, para as estruturas no banco de dados e vice-versa.

O que é perdido em flexibilidade é ganho em simplicidade, já que o sistema está baseado em um modelo concreto. Já que a transferência de dados do banco de dados para um documento XML segue um único modelo, XSL é geralmente aplicado em produtos (*middleware*) dirigidos a modelo para prover a flexibilidade encontrada em sistemas dirigidos a *templates*.

---

[1] Incluindo no lugar de tags que foi usado como parâmetro de uma declaração SELECT.
[2] Estruturas de repetição normalmente encontradas em linguagens de programação.
[3] Estruturas de decisão normalmente encontradas em linguagens de programação.
[4] HyperText Transfer Protocol; Protocolo de Transferência de Hypertexto.

Dois modelos para visualizar dados em um documento XML são comuns: o *modelo de tabela* e o *modelo de objeto de dados específicos*[DM98]. Outros modelos de dados em um documento XML também são possíveis. Por exemplo, pelo uso de ID e atributos IDREF, um documento XML pode ser usado para representar um gráfico dirigido. Porém, estes modelos ainda não estão amplamente suportados pelos middlewares disponíveis.

C.1.2.1 – Modelo de Tabela

O mapeamento entre XML e tabelas relacionais é um pouco mais complicado que o mapeamento entre XML e objetos, pois são dois modelos diferentes. A figura a seguir mostra que não existe um relacionamento direto entre XML e tabelas (diferente de XML e objetos), pois agrupamentos relacionais adicionais são necessários para criar um documento XML .

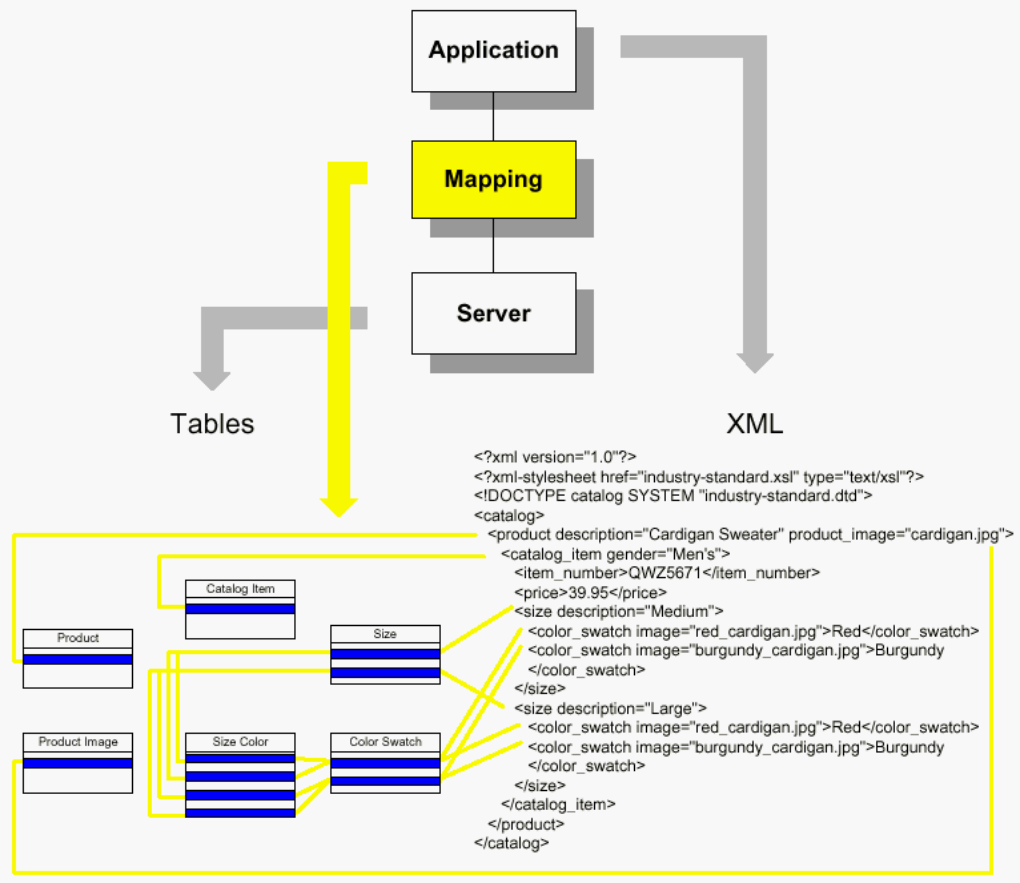


Figura 60 – Diagrama de mapeamento entre tabelas e XML

O modelo de tabela é usado por muitos dos pacotes de *middleware* para transferir dados entre um documento XML e banco de dados relacionais. Ele modela um

documento XML como uma única tabela ou conjunto de tabelas. Quer dizer, a estrutura do documento XML deve ser semelhante ao mostrado a seguir, onde o elemento <bancodedados > não existe no caso de única tabela:

```
<bancodedados>
   <tabela>
      <linha>
         <coluna1>...</coluna1>
         <coluna2>...</coluna2>
         ...
      </linha>
      ...
   </tabela>
   ...
</bancodedados>
```

Figura 61 – Exemplo do modelo de tabela.

O termo "tabela" pode ser interpretada como sendo um único conjunto de resultado (ao transferir dados do banco de dados para XML) ou uma única tabela ou uma visão de dados atualizável (ao transferir dados XML para o banco de dados).

Se dados de mais de um conjunto de resultados é desejado (ao transferir dados do banco de dados) ou o documento XML contém elementos profundamente aninhados que são necessários serem representados por um conjunto de tabelas (ao transferir dados ao banco de dados), então a transferência simplesmente não é possível.

C.1.2.2 – Modelo de Objeto de Dados Específicos

O mapeamento entre XML e objetos é a forma mais simples de mapear, já que o modelo de objetos e o modelo XML são muitos similares.

De fato, se for analisado um documento XML, terá uma árvore que é uma estrutura de dados comum para modelos de objetos. A figura a seguir mostra que existe um relacionamento de um para o outro de cada objeto relatado e seu correspondente XML.

Neste diagrama, o mapeamento é mostrado usando um pequeno exemplo somente para demonstrar essa relação.

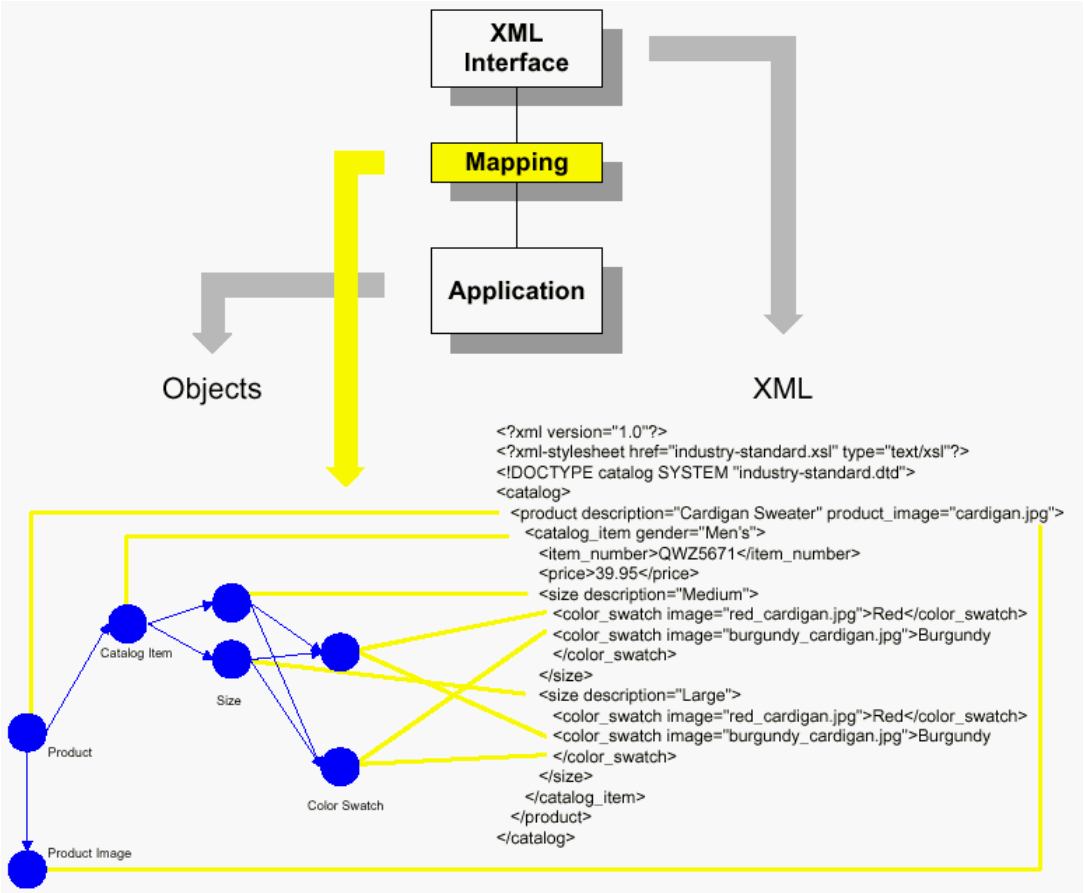


Figura 62 – Diagrama de mapeamento entre objetos e XML

Esse modelo é comum para mapear dados em um documento XML em uma *árvore de objetos de dados específicos* nos quais tipos de elemento geralmente correspondem a objetos e modelos de conteúdos, atributos, e PCDATA correspondem a propriedades.

Este modelo mapeia diretamente a bancos de dados orientados a objeto e hierárquicos, e pode ser mapeado a bancos de dados relacionais que usam técnicas de mapeamento objeto/relacional tradicional ou visões de objeto do SQL 3.

Note que este modelo não é o Modelo de Objeto de Documento (DOM), pois o DOM modela o próprio documento, não os dados no documento, e é usado para construir sistemas de gerenciamento de conteúdo em cima de bancos de dados relacionais (a ser visto mais adiante neste capítulo).

Por exemplo, o documento de venda mostrado anteriormente poderia ser visto como uma árvore de objetos de cinco classes (`Venda`, `PedidoVenda`, `Cliente`, `Linha`, e `Item`), como mostrado no diagrama a seguir:

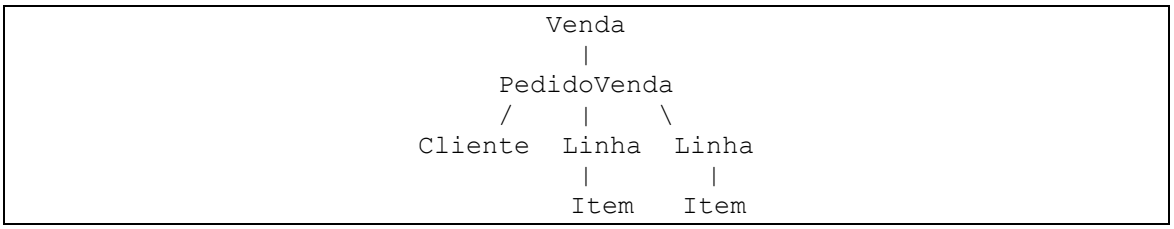


Figura 63 – Exemplo do modelo de objeto.

Quando se modela um documento XML como uma árvore de objetos dados específicos, não há nenhuma exigência que elementos necessariamente correspondam a objetos. Por exemplo, se um elemento contém só PCDATA, como o elemento CliNom no documento de pedido de vendas, o mesmo pode ser visto como uma propriedade porque, como uma propriedade, contém um único valor escalar.

Semelhantemente, às vezes é útil modelar elementos com conteúdo misto ou elementar como sendo propriedades. Um exemplo disto é o elemento descrição no documento de pedido de vendas: embora foi misturado conteúdo na forma de XHTML, é mais útil para ver a descrição como uma única propriedade porque seus pedaços de componente são sem sentido por eles próprios.

C.3 – Gerando DTDs a partir de Schemas de BDs e Vice-Versa

Uma pergunta comum ao transferir dados entre documentos XML e um banco de dados é como gerar um *schema* DTDs XML a partir do banco de dados e vice-versa.

Em resumo, esta é uma operação bastante direta. Deve ser notado, que esta geralmente é uma operação que é executada uma única vez, como a maioria das aplicações, especialmente aplicações verticais que trabalham com um conjunto conhecido de *schemas DTD/relacional*.

As exceções são ferramentas que armazenam documentos XML randômicos em bancos de dados relacional ou aquele que expões dados relacionais como documentos XML. Na verdade, não é absolutamente claro a utilidade de DTDs nesse caso .

Por exemplo, o procedimento a seguir (um tanto simplista) gera um schema relacional de um DTD[1]:

[1] Este mapeamento foi explicado com detalhes no Capítulo 4.

1. Para cada tipo de elemento de conteúdo misto ou elementar, crie uma tabela e uma coluna que seja chave primária.
2. Para cada tipo de elemento com conteúdo misto, crie uma tabela separada para armazenar o PCDATA, unido à tabela pai pela chave primária do pai.
3. Para cada atributo de único valor daquele tipo de elemento, e para cada tipo de elemento filho que ocorre isoladamente com conteúdo PCDATA puro, crie uma coluna naquela tabela. Se o tipo de elemento filho ou atributo é opcional, faça uma coluna nula.
4. Para cada atributo multivalorado e para cada tipo de elemento de filho que ocorre várias vezes com conteúdo PCDATA puro, crie uma tabela separada para armazenar valores, unido à tabela pai pela chave primária do pai.
5. Para cada tipo de elemento filho com conteúdo misto ou elementar, una os elementos da tabela pai aos elementos da tabela filho com a chave primária do pai.

O procedimento a seguir (também simplista) gera um DTD a partir de um *schema* relacional:

6. Para cada tabela, crie um elemento.
7. Para cada coluna em uma tabela, crie um atributo ou um elemento filho PCDATA puro.
8. Para cada relacionamento chave primária/chave estrangeira na qual uma coluna de uma tabela relaciona a uma chave primária, crie um elemento filho.

Infelizmente, há várias desvantagens nestes procedimentos. Por exemplo, não há como predizer definitivamente tipos de dados ou comprimentos de coluna a partir do DTD. Qualquer predição, como o que é feito lendo um documento de amostra, pode ser posto abaixo colocando simplesmente dados de outro "tipo" ou um caractere de mais comprimento em um documento. (A solução a longo prazo para este problema é o uso de tipos de dados em documentos de schema XML.)

Semelhantemente, ao gerar um DTD de um schema relacional, não há como predizer a ordem na qual elementos filhos deveriam "acontecer" ou se uma coluna, como um identificador de linha interno ao banco de dados, deveria ser transferida ou não. Em ambos os casos, colisões são possíveis.

Apesar destas desvantagens, estes algoritmos provêem ainda um ponto de partida útil para gerar DTD a partir de schema relacional e vice-versa

Bibliografia

[AG89] Amar Gupta. *Integration of Information Systems : Bridging Heterogeneous Databases (IEEE Press Selected Reprint Series)*. Inst. of Eletrical. Dezembro/1989.

[AMA98] Ahmed Elmagarmid, Marek Rusinkiewicz, Amit Sheth. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann Publishers. Outubro/1998.

[AR00] Akmal B. Chaudhri, Roberto Zicari. *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML*. John Wiley & Sons. Setembro/2000.

[BM00] Benoit Marchal. *XML: Conceitos e Aplicações*. Berkeley. Setembro/2000.

[CD88] C. J. Date. *Banco de Dados (Tópicos Avançados*). Campus. Setembro/1988.

[CD91] C. J. Date. *Introdução a Sistemas de Banco de Dados*. Campus. Fevereiro/1991.

[DD98] Dan Chang, Dan Harkey. *Client/Server Data Access With Java and XML*. John Wiley & Sons. Setembro/1998.

[DM98] David Megginson. *Structuring XML Documents*. Prentice Hall Computer Books. Janeiro/1998.

[GD99] George M. Doss. *CORBA Developer's Guide with XML*. Wordware Publishing. Agosto/1999.

[HK99] Henry F. Korth. *Sistema de Banco de Dados*. Makron. Maio/1999.

[HTR00] G. Hussain Chinov, Tyna Hull, Robi Sen, Hussain Chinov. *XML for EDI : Making E-Commerce a Reality*. Morgan Kaufmann Publishers. Novembro/2000.

[JM00] JP Morgenthal. *Enterprise Applications Integration with XML and Java*. Prentice Hall. Julho/2000.

[NP00] Natanya Pitts-Moultis. *XML – Black Book. Solução e Poder*. Makron Books. Fevereiro/2000.

[SE99] Simon St. Laurent, Ethan Cerami. *Building XML Applications*. McGraw Hill Professional Publishing. Maio/1999.

[SH00] Steven Holzner. *Inside XML*. New Riders. Novembro/2000.

[SM00] Steve Muench. *Building Oracle XML Applications*. O'Reilly & Associates. Outubro/2000.

[SM00] Sean MacGrath. *XML – Aplicações Práticas*. Campus. Junho/2000.

[SP00] Sams Publishing. *Working With Microsoft Sql Server 2000 and XML*. Sams. Dezembro/2000.

[SPD99] Serge Abiteboul, Peter Buneman, Dan Suciu. *Data on the Web : From Relations to Semistructured Data and Xml (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers. Outubro/1999.

[SS99] Stephen T. Mohr, Stephen F. Mohr. *Designing Distributed Applications With XML : ASP IE5 LDAP and MSMQ*. Wrox Press. Maio/1999.

[VA00] Vipul Kashvap, Amit Sheth. *Information Brokering over Heterogeneous Digital Data : A Metadata-Based Approach (The Kluwer International Series on Advances in Database Systems)*. Kluwer Academic Pub. Maio/2000.

[VS88] Valdemar Waingort Setzer. *Banco de Dados*. Edgard Blucher. Janeiro/1988.

[WT00] Wrox Author Team. *Professional Metadata W/ DTD's , XML Schemas, Topic Maps, RDF, Webdav, XML Servers*. Wrox Press. Julho/2000.

[XML] *Extensible Markup Language (XML) 1.0*: http://www.w3.org/TR/1998/REC-xml-19980210. *Pesquisado em 10/10/2000*

[SOAP] *Simple Object Access Protocol (SOAP) Version 1.1*: http://www.w3.org/TR/SOAP. *Pesquisado em 10/10/2000.*

[XMLNS] *Namespaces in XML*: http://www.w3.org/TR/1999/REC-xml-names-19990114. *Pesquisado em 10/10/2000.*

[URI] *Uniform Resource Identifiers (URI): Generic Syntax*: http://www.ietf.org/rfc/rfc2396.txt. *Pesquisado em 01/12/2000.*

[ISO8601] *ISO 8601: Representations of dates and times:* http://www.iso.ch/markete/8601.pdf. *Pesquisado em 01/12/2000.*

[HTTP] *Hypertext Transfer Protocol—HTTP/1.1:* http://www.ietf.org/rfc/rfc2616.txt. *Pesquisado em 08/12/2000.*

[XMLMIME] *XML Media Types:* http://www.ietf.org/rfc/rfc2376.txt. *Pesquisado em 10/10/2000.*

[MULTIPART] *The MIME Multipart/Related Content-type:* http://www.ietf.org/rfc/rfc2387.txt. *Pesquisado em 01/10/2000.*

[MIME1] *MIME Part One: Format of Internet Message Bodies:* http://www.ietf.org/rfc/rfc2045.txt. *Pesquisado em 01/12/2000.*

[MIME2] *MIME Part Two: Media Types:* http://www.ietf.org/rfc/rfc2046.txt. *Pesquisado em 01/12/2000.*

[MIME3] *MIME Part Three: Message Header Extensions for Non-ASCII Text:* http://www.ietf.org/rfc/rfc2047.txt. *Pesquisado em 01/12/2000.*

[MIME4] *MIME Part Four: Registration Procedures:* http://www.ietf.org/rfc/rfc2048.txt. *Pesquisado em 01/12/2000.*

[SMIME] *S/MIME Version 3 Message Specification:* http://www.ietf.org/rfc/rfc2633.txt. *Pesquisado em 01/12/2000.*

[CID] *Content-ID and Message-ID Uniform Resource Locators:* http://www.ietf.org/rfc/rfc2111.txt. *Pesquisado em 08/12/2000.*

[XDR] *XML-Data Reduced (XDR):* http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm. *Pesquisado em 05/02/2001.*

[XSD1] *XML Schema Part 1: Structures:* http://www.w3.org/TR/xmlschema-1. *Pesquisado em 05/02/2001.*

[XSD2] *XML Schema Part 2: Data types:* http://www.w3.org/TR/xmlschema-2. *Pesquisado em 05/02/2001.*

[XP] *XML Protocol:* http://www.w3.org/2000/xp/. *Pesquisado em 10/10/2000.*