Trabalho de Graduação

Armazenamento de Dados e Objetos Virtuais utilizando o Suporte Nativo a XML do Oracle 10g

Aluna: Nara de Arruda Falcão (naf@cin.ufpe.br)

Orientador: Fernando da Fonseca de Souza (fdfd@cin.ufpe.br)

Desenvolvido no âmbito do projeto AMADeUs-MM

Processo nº 507487/2004-4, edital CNPq nº 014/2004

Fomento Tecnológico

Recife, janeiro de 2006

Agradecimentos

Á minha família, namorado e amigos que agüentaram meu stress durante todo o curso.

Resumo

Com o novo cenário de aplicações voltado para a Web, a busca por aplicações mais interativas tem incentivado cada vez mais a construção de ferramentas de manipulação de mídias e objetos virtuais para exibição na Web. Este trabalho tem como objetivo a especificação e implementação de um módulo de gerenciamento de mídias e objetos virtuais. Para chegar a este módulo serão discutidas as possíveis formas de armazenamento de mídias e ambientes virtuais utilizando o suporte nativo a XML do Oracle 10g.

Abstract

With the new scenario of applications focused on the Web, the search for more interactive applications has stimulated the construction of tools to manipulate medias and virtual objects for exhibition in the Web. This paper is aimed at specifying and implementing a module to manage medias and virtual objects. To reach this target, it will be discussed the different forms to storage medias and virtual environments using the native XML support of the Oracle 10g.

Sumário

Lista de Figuras

Lista de Quadros

1. Introdução

O surgimento da internet revolucionou a sociedade. As grandes aplicações vêm mudando, já a algum tempo, de desktop para a Web. Com o novo cenário de aplicações voltado para a Web, a busca por aplicações mais interativas tem incentivado cada vez mais a construção de ferramentas de manipulação de mídias e objetos virtuais para exibição na Web

Devido a importância da troca de informações entre as aplicações, a eXtensible Markup Language (XML) vem se destacando uma vez que apresenta suporte a dados semi-estruturados e possibilita a declaração mais precisa do conteúdo das aplicações. XML está transformando a Web para sua próxima geração. A primeira geração da Web mudou significativamente várias indústrias, no entanto o impacto desta próxima geração poderá ser mais profundo. A principal força de XML é sua capacidade de representar dados tanto estruturados como não-estruturados.

Este trabalho tem como objetivo a especificação e implementação de um módulo de gerenciamento de mídias e objetos virtuais. Para chegar a este módulo serão discutidas as possíveis formas de armazenamento de mídias e ambientes virtuais utilizando o suporte nativo a XML do Oracle 10g.

1.1. Motivação

O motivo para a realização desde trabalho deve-se ao uso cada vez maior de mídias e objetos virtuais na Web. A disseminação da internet tem incentivado cada vez mais a construção de ferramentas de manipulação de mídias para exibição na Web. Entretanto os ambientes atuais apresentam algumas limitações. Uma delas é a inexistência de mecanismos de reutilização de objetos de mundos virtuais de um banco de dados (BD).

A utilização de um sistema de gerenciamento de banco de dados (SGBD) com suporte a XML é de extrema importância para o gerenciamento de mídias e objetos virtuais.

1.2. Objetivo

O objetivo deste trabalho é a especificação e a implementação de um modelo de gerenciamento de mídias e objetos de realidade virtual, de modo que simplifique o processo de armazenamento e exibição dos mesmos na Web.

Desta maneira, este trabalho pretende especificar todos os passos necessários desde o armazenamento da mídia até sua execução na Web através de um browser.

Serão discutidas as tecnologias envolvidas para a manipulação da mídia através de documentos XML, bem como os recursos que o SGBD Oracle 10g dispõe para facilitar a programação utilizando XML.

Como resultado deste trabalho será construído um modelo de gerenciamento de mídias e objetos de realidade virtual denominado MidiaWEB. Como estudo para homologação do MidiaWEB, será construído um sistema que permitirá a exibição de mídias e mundos virtuais a partir de objetos armazenados em um SGBD. Testes serão realizados de modo a mostrar a viabilidade do modelo proposto.

1.3. Estrutura do trabalho

Além deste capítulo introdutório que contém o contexto do tema proposto, a justificativa e os objetivos, o trabalho é composto por mais cinco capítulos descritos a seguir.

Capítulo 2 – XML. Discute o que é XML e sua importância na internet, suas características e as principais diferenças entre XML e HTML. Este capítulo fala ainda da sintaxe de XML e suas restrições, diz o que são DTD e XML Schema e revela qual a diferença entre um XML válido e um XML bem definido. Por fim são abordadas as tecnologias DOM, SAX, XPath e XQuery.

Capítulo 3 – SGBD XML Nativos. Discute o que é um SGBD XML nativo e suas características. Fala sobre as possíveis arquiteturas para banco de dados XML nativo. Apresenta ainda um exemplo de um banco XML nativo, o Berkeley DB XML, e fala do suporte nativo do Oracle 10g a XML.

Capítulo 4 - Representação de Mídias e Objetos de Realidade Virtual. Este capítulo trata das tecnologias para representação dos objetos de realidade virtual e de mídias. Primeiramente aborda VRML, suas características e sintaxe. É focalizada ainda a tecnologia X3D e suas diferenças e semelhanças a VRML.

Capítulo 5 – Estudo de Caso/Protótipo. Apresenta o modelo de gerenciamento de mídias e objetos virtuais implementado. Descreve o sistema de execução de mídias e objetos virtuais em um browser na Web.

Capitulo 6 – Conclusão. Destaca as contribuições do trabalho e apresenta sugestões para trabalhos futuros.

2. EXtensible Markup Language (XML)

A linguagem Extensible Markup Language (XML) [8] inicialmente foi projetada para atender as necessidades das publicações em larga escala de textos na Web. Ela é um formato de texto simples e muito flexível. XML ganha destaque na importância da troca de uma grande variedade de dados em outros lugares e na Web. XML evita os problemas mais comuns em projetos de linguagens; ela é extensível, independente de plataforma e dá suporte à internacionalização e à localização.

XML é uma linguagem de marcação muito semelhante a HTML. Apesar de serem parecidas, estas duas linguagens têm objetivos diferentes. XML foi criada para descrever dados e dizer o que o dado é, enquanto que HTML foi projetada para visualizar o dado com destaque na aparência visual do dado. Em XML o usuário é encarregado de criar suas próprias *tags* (marcadores), o que significa que elas não são pré-definidas. XML é modular, isto quer dizer que ela permite que o usuário defina um novo formato de documento reutilizando ou combinando vários formatos. Para descrever o dado, XML utiliza um Document Type Definition (DTD) ou um XML Schema, tornando-o auto-descritivo [33].

XML foi desenvolvida pela World Wide Web Consortium (W3C), empresa responsável na definição da área gráfica da internet. O aparecimento de XML teve como propulsor a necessidade de superar as limitações de HTML, linguagem esta que é padrão nas páginas da Web. XML é definida como o formato universal para dados estruturados na Web. Cabe a linguagem definir regras que possibilitam escrever documentos de modo que sejam visíveis ao computador [31].

2.1. Principal Diferença entre HTML e XML

XML não é um substituto para HTML. Estas linguagens foram projetadas para atingir objetivos diferentes. XML foi criada para descrever o dado e como ele é, enquanto que HTML foi projetada para exibir o dado e como será sua aparência. Em resumo, HTML exibe dados, já XML descreve dados [33].

A principal semelhança entre HTML e XML está no fato de ambas utilizarem *tags*. Cada *tag* possui duas partes, uma que dá inicio ao comando e outra que termina o comando. A diferença entre as duas linguagens é que HTML é mais tolerante a erros. Um exemplo disso seria o caso de abrir uma *tag* e não fechá-la. Em HTML isto é permitido e as informações seriam exibidas normalmente. Mas em XML isto é considerado um erro. Aplicação simplesmente pára. Apesar de

este exemplo parecer uma desvantagem em relação a XML, existe uma enorme recompensa pela extensibilidade de XML [31].

As *tags* nas duas linguagens também têm objetivos diferentes. Em XML as *tags* são usadas para definir blocos de dados. A *tag* <*b*></*b*> por exemplo, pode significar barco, bar, ou outra coisa, ficando a critério do programador. Já em HTML estas mesmas *tags* são utilizadas unicamente para indicar que o texto está em negrito. XML usa os marcadores apenas para delimitar os trechos de dados, deixando sua interpretação completamente a cargo da aplicação que os lê [31].

2.2. Características de XML

As principais características de XML estão descritas a seguir [1]:

Extensível - XML se torna extensível quando se tem a possibilidade de criar marcadores de um modo arbitrário. Isto permite que uma estrutura de um documento XML se adapte a praticamente qualquer situação específica;

Documentos auto-descritivos - Fazer a interpretação e manipulação de um documento XML é relativamente fácil. Esta característica permite que uma busca efetuada na Web possa levar em conta o significado (contexto) dos dados, ao invés de simplesmente se basear na combinação de palavras-chaves;

Múltiplas Perspectivas - É possível obter múltiplas perspectivas de um mesmo documento XML. Isto ocorre uma vez que o conteúdo do documento está separado da sua apresentação; e

Livre de licenças e independente de plataforma - XML é um padrão aberto. Os documentos XML são independentes das aplicações e dos sistemas operacionais, por exemplo.

2.3. Sintaxe de XML

As regras de sintaxe de XML são fáceis [33]. Elas são simples e bem restritas, o que as torna fáceis de aprender e usar. A sintaxe dos documentos XML é auto-descritiva e simples. Observe o exemplo a seguir no Quadro 1.

Quadro 1 – Exemplo de XML

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <filme>
3 <titulo>The Matrix</titulo>
4 <diretor>The Wachowski Brothers</diretor>
5 <atorPrincipal>Keanu Reeves</atorPrincipal>
6 <genero>Ficção</genero>
7 </filme>
```

A linha 1 do Quadro 1 é responsável em definir a versão de XML e qual o tipo de codificação de caractere que será usada no documento. No exemplo anterior pode-se perceber que este documento XML utiliza a especificação 1.0 de XML e que usa o conjunto de caracteres da ISO-8859-1 (Latin-1 / West European). Na segunda linha do Quadro 1 é descrito o elemento raiz do documento, neste caso um filme. As linhas 3, 4, 5 e 6 descrevem quatro elementos filhos da raiz. São eles: titulo, diretor, ator principal e gênero, respectivamente. Por fim, a última linha define o término do elemento raiz “filme”.

2.3.1 Restrições da Sintaxe

Diferentemente de HTML, em XML todos os elementos devem conter uma *tag* de fechamento, não sendo permitido sua omissão. Observe agora o código no Quadro 2.

Quadro 2 - Elemento Válido: HTML e XML

```
1 <b>Valido em HTML
2 <b>Valido em XML</b>
```

A primeira linha do Quadro 2 exibe um elemento válido para HTML, mas inválido para XML. Para este exemplo não fugir das regras da sintaxe de XML, é necessário finalizar o elemento com a *tag* de fechamento como mostra a linha 2.

A sintaxe de XML é sensível à maiúscula / minúscula, o que a distingue mais uma vez da sintaxe de HTML. Em XML a tag <Tamanho> é diferente da *tag* <tamanho>. Os marcadores que

indicam o início de um elemento e os marcadores que indicam seu fim devem respeitar a grafia maiúsculo / minúsculo. Um exemplo válido é exibido no Quadro 3.

Quadro 3 – Exemplo Válido de *tag* XML

```
<tamanho>10cm</tamanho>
```

Quanto ao aninhamento dos elementos, a sintaxe de XML é bem rigorosa. Todos os elementos devem estar devidamente aninhados. Deve existir somente um único par de marcadores para definir o elemento raiz e todos os elementos filhos devem estar contidos dentro deste. Cada elemento pode conter sub-elementos, mas deve tomar cuidado para aninhá-los corretamente com seus pais.

Em documentos XML é possível ter atributos como em HTML. No entanto, em XML o valor do atributo deve obrigatoriamente vir entre aspas duplas. No exemplo do Quadro 4 é possível observar como é feita a declaração do atributo.

Quadro 4 – Declaração de Atributo

```
<?xml version="1.0"?>
<consulta hora="16:30">
<paciente>José Alencar</paciente>
<medico>Angelina Ramos</medico>
<data>15/01/2006</data>
</consulta>
```

2.4. Document Type Definition (DTD) e XML Schema

Em um documento XML podem estar contidos vários tipos de informações. Isto significa que para qualquer grupo de usuários podem existir várias linguagens escritas [11,21]. Por exemplo, usuários médicos utilizariam XML para armazenar diagnósticos dos seus pacientes. Já uma loja utilizaria XML para armazenar dados das vendas, dos produtos e clientes. Para cada tipo de organização pode-se utilizar XML para diferentes fins.

Para definir cada um dos usos de XML utilizam-se linguagens próprias ou metalinguagens. Estas linguagens são definidas especificando-se quais marcadores podem ou não ser utilizados, qual deve ser a ordenação desses marcadores dentro do documento, ou ainda quais os atributos

que podem ou devem ser especificados em cada marcador. Para que isto ocorra, existem duas metalinguagens capazes de definir estas linguagens específicas, a Definition Type Document (DTD) [6] e XML Schema [29].

DTD seria uma espécie de glossário. Sua sintaxe é especial e diferente da de XML. Para evitar o aprendizado de uma nova sintaxe, criou-se então o XML Schema. A sintaxe de XML Schema é igual à de XML. Em comparação a DTD, XML Schema utiliza mais recursos e pode ser mais complicado especificar uma linguagem. Sua vantagem é que não é preciso aprender uma nova sintaxe em particular.

DTD e XML Schema comprovam a integridade dos dados em qualquer instante. Elas garantem que em qualquer documento XML os dados são válidos, assegurando que uma aplicação que não possui internamente uma descrição dos dados possa validá-los mesmo assim.

2.5. XML Bem Formado e XML Válido

Dizer que um documento XML é “bem formado” significa que sua sintaxe está correta. Já os documentos validados por uma DTD são chamados de XML válido [33]. A especificação de XML define que nenhum programa deve continuar a processar um documento XML caso ele não seja bem formado, ou seja, que exista algum erro de sintaxe. O motivo para tal procedimento é devido ao fato de um “software XML ser fácil de escrever e que todos os documentos XML devem ser compatíveis“ [33].

Em HTML é permitido criar documentos com vários erros. Esse é o principal motivo para que os *browsers* HTML sejam tão incompatíveis e grandes, pois cada um deles tem seus próprios métodos de descobrir como um documento deve ser exibido quando forem encontrados erros em HTML. Isto não é possível em XML.

2.6. XPath e XQuery

Para fazer uma consulta a documentos XML pode-se utilizar duas linguagens: XPath e XQuery [20,22]. Essas linguagens são capazes de percorrer o documento XML, definir predicados de seleção e retornar como resultado dados no formato XML.

A primeira recomendação da W3C para consultas a dados XML foi a XPath. Esta linguagem funciona navegando pela estrutura do documento e recuperando os pedaços do documento que casem com uma expressão de navegação. A sintaxe da XPath se assemelha muito à sintaxe de navegação nas pastas (diretórios) do DOS [14] ou do Unix [23].

Apesar de satisfatória em diversas consultas, XPath apresenta algumas limitações. Não é possível fazer junções entre dados XML e ainda é impossível obter resultados com uma estrutura diferente da existente no documento. Esta última limitação deve-se ao fato de expressões XPath retornarem somente partes de um documento XML.

Para corrigir essas limitações da linguagem Xpath, a W3C criou a XQuery. Ela “é uma linguagem de consulta declarativa para dados XML” [22] parecida com a linguagem SQL. Com ela, é possível realizar pesquisas sobre um ou mais documentos e pode-se ainda construir novos documentos a partir dos resultados de uma seleção. XQuery é ideal para pesquisa, porém ela não dá suporte a *updates* ou *inserts*.

O Quadro 5 mostra dois exemplos de comandos sobre um documento XML em Xpath (a) e em XQuery (b).

Quadro 5 – Exemplos de Comandos Xpath e XQuery

```
1 /biblioteca/livro/titulo
2
//livro/autor[nome=’Maria´]/titulo
3 //livro[@versao > 1]/*/@nome
```

(a)

```
1 for $art in /biblioteca/livro
2 let $v := $art/@versao * 10
3 where $art/autor/nome = “Maria”
4 return {$art/titulo, $v}
```

(b)

Em (a) do Quadro 5 é possível verificar a existência de três expressões XPath executadas sobre um documento XML. A linha 1 indica que o que será recuperado é o titulo do livro. jJ na linha 2 é acrescentada uma condição onde o nome do autor do livro deve ser ‘Maria’. Finalmente, a última linha busca atributos (indicados pelo caractere @) nome de todos os elementos filhos do elemento livro (indicado pela presença do *) cuja versão seja maior que 1.

Um exemplo de uma consulta em XQuery é mostrada em (b) ainda no Quadro 5. Esta consulta retorna títulos de livros onde o autor seja igual a ‘Maria’, bem como a versão do livro multiplicada por 10. Com este exemplo, é possível enxergar a capacidade de XQuery de gerar estruturas diferentes das presentes no documento XML.

2.7. DOM e SAX

A W3C especificou dois mecanismos para trabalhar e acessar documentos XML. São normas que dizem aos programadores como devem ser acessados os documentos. Estes mecanismos se chamam Simple API for XML (SAX) [17] e Document Object Model (DOM) [27,26].

"SAX utiliza-se para fazer um percurso da seqüência dos elementos do documento XML e DOM implica a criação de um organograma na memória que contem o documento XML, e com ele na memória podemos fazer qualquer tipo de percurso e ações com os elementos que quisermos" [11].

DOMé uma tecnologia que permite às aplicações fazerem o processamento de dados XML. Para que isto ocorra, utiliza-se um conjunto de métodos pré-definidos que fazem a manipulação de documentos XML de acordo com sua API (Application Program Interface). DOM faz a representação do documento XML através de uma estrutura hierárquica de árvores. Através dos métodos da API é possível percorrer os nós da árvore, ou seja, os elementos, seus atributos e seus valores. Um método de DOM pode ser ativado dentro de uma programa JavaScript por exemplo.

3. SGBD XML Nativo

Após a definição do que é um documento XML e de suas aplicabilidades, será discutido como fazer para armazená-los [12,28]. Existem duas formas básicas de guardar um documento XML. Uma alternativa seria armazená-lo num sistema de arquivo ou em um campo do tipo BLOB (tipo de dados binário) de um banco de dados relacional. Para essas possibilidades o usuário deve se contentar em ter funcionalidades XML limitadas. Outra alternativa é armazenar os documentos XML em um banco de dados XML nativo.

Para documentos simples o modo mais fácil de armazenamento seria em sistema de arquivos. Obviamente, neste tipo de armazenamento, consultas no corpo do documento XML são impossíveis, uma vez que este sistema não distingue os marcadores do resto do texto e não entendem o uso de entidade. Juntamente com ferramentas auxiliares de controle de versões, como o CVS (Concurrent Version Control) [5], pode-se ter um controle simples de transação.

Outra possibilidade para o armazenamento de documentos XML seria armazená-los em banco de dados relacionais com suporte a BLOB (Binary Large Object). Em relação à alternativa anterior, esta já apresenta um número enorme de vantagens por se tratar de um SGBD. Pode-se citar: controle de transação, segurança, acesso multi-usuário, entre outros. Os SGBD ainda apresentam ferramentas de pesquisa em textos. Algumas dessas ferramentas já apresentam suporte a buscas em documentos XML, sem tratá-los como um texto normal. Com o uso do SGBD, cada usuário pode ainda criar sua própria indexação de documentos XML.

Apesar de interessante, esta última alternativa ainda possui limitações. Nesse contexto é que entram os SGBD XML. Eles são sistemas especializados no armazenamento de documentos XML, apresentando as mesmas vantagens que qualquer SGBD, ou seja, suporte a controle de transações, controle de concorrência, recuperação pós-falha, entre outras. A única diferença é que o modelo interno deles é baseado em XML e não em modelos como o relacional.

Existem dois tipos de SGBD XML [7]. O primeiro é o que “permite” XML, que é um SGBD relacional ou orientado a objetos que foi expandido para dar suporte a dados XML. Neste tipo de sistema, há sempre uma conversão do documento XML para a estrutura nativa do SGBD. O segundo tipo é SGBD XML nativo, o qual indexa documentos XML diretamente e armazena todo o documento e seus elementos relacionados. Dependendo da aplicação, o SGBD XML nativo pode apresentar melhor performance que os SGBD que apenas sofreram uma expansão.

SGBD XML nativo dão suporte a linguagens de consulta XML e ainda podem limitar a pesquisa apenas para certas partes do documento. Eles preservam ainda a ordem do documento,

instruções de processamento, comentários, uso de entidade e algumas vezes a seção CDATA. Tudo isso é algo que SGBD que “permitem” XML não o fazem.

Nas seções seguintes será discutido o que é um SGBD XML nativo, bem como suas características e suas possíveis arquiteturas. Como exemplo de SGBD XML nativos serão estudados os sistemas Berkeley DB XML e o Oracle 10g. O Oracle é um produto comercial e que apresenta vários recursos disponíveis. Ele será discutido com mais detalhes por se tratar do foco deste trabalho. O Berkeley também é um produto comercial, seu suporte a XML ganhou destaque entre as principais comunidades de usuários de SGBD XML nativo por ser compatível com quase todas as aplicações. Existe uma grande variedade de materiais disponível na internet sobre o Berkeley DB XML.

3.1. SGBD XML Nativo

O termo “SGBD XML nativo” ganhou nome com a campanha de mercado do Tamino [19], um SGBD XML nativo da Software AG. Pelo sucesso desta campanha, o termo passou a ser utilizado por outras empresas que produzem produtos similares. Por ser um termo de marketing não existe uma definição técnica sólida para ele.

Uma possível definição técnica para o termo é que um SGBD XML nativo deve seguir três critérios [30]:

- Define um modelo lógico para um documento XML e armazena e recupera documentos de acordo com este modelo. No mínimo, o modelo deve incluir elementos, atributos, PCDATA, e ordem do documento;
- Deve ter como unidade (lógica) fundamental de armazenamento o documento XML, assim como um banco de dados relacional tem uma linha na tabela como unidade; e
- Não é necessário que se tenha algum modelo físico de armazenamento particular. Isto significa que ele pode ser construído em cima de um SGBD hierárquico, ou relacional, ou orientado a objetos, por exemplo.

O primeiro item da definição é igual às definições de outros SGBD, focando o modelo usado pelo sistema. O SGBD é especializado em armazenar dados XML e também armazena todos os componentes do modelo XML intactos. É importante destacar que um SGBD XML nativo pode armazenar mais informações do que está contido no modelo que ele usa. Por exemplo, ele pode dar suporte a consultas baseadas em modelos de dados Xpath, mas armazena documentos como texto.

O segundo ponto da definição assegura que a unidade de armazenamento fundamental de um SGBD XML nativo é o próprio documento XML. Em outras palavras, documentos entram e documentos saem. Ainda é possível que um SGBD XML nativo possa atribuir esse papel para os fragmentos dos documentos.

O último item da definição afirma que o formato do armazenamento do dado não é importante. É verdade, e é análogo dizer que o formato físico do armazenamento usado pelo SGBD relacional não está relacionado a se o SGBD é relacional. Em resumo, isto significa que um SGBD XML nem precisa ser um sistema autônomo.

Deve ficar claro ainda que, pela definição, SGBD XML nativos não representam um modelo de baixo nível, e não têm a intenção de substituir os SGBD já existentes. Eles simplesmente são uma ferramenta que visa dar assistência ao desenvolvedor provendo de forma robusta a manipulação e o armazenamento de documentos XML [12].

3.2. Características de SGBD XML Nativo

Apesar de nem todos os SGBD XML nativo serem iguais, entre eles existem várias características em comum que motivam uma discussão básica sobre elas. A seguir será abordada cada uma delas.

3.2.1. Armazenamento de XML

SGBD' XML nativo armazenam documentos XML como uma unidade e criam um modelo que é muito próximo a XML ou está relacionado a alguma tecnologia de XML, como DOM. Esse modelo inclui níveis arbitrários de aninhamento complexo, assim como suporte completo para conteúdos misturados e dados semi-estruturados. Este modelo é automaticamente mapeado pelo SGBD XML nativo em um mecanismo de armazenamento subjacente. O mapeamento usado assegura que o modelo do dado específico de XML é mantido. Uma vez que o dado foi armazenado, deve-se continuar a usar as ferramentas de SGBD XML nativo se for esperado ter uma representação útil do dado. Por exemplo, se alguém estiver utilizando um SGBD XML nativo implementado sobre um SGBD relacional, o acesso às tabelas usando diretamente SQL não deve ser tão útil como o esperado. A razão para isso é simplesmente porque um dado que retornar será o modelo do documento XML e não as entidades de negócios que o dado representa. O modelo da entidade de negócio existe juntamente com o domínio do documento XML. Ele não conterá o domínio do sistema de armazenamento de dados subjacente. Para trabalhar com o dado, é necessário trabalhar com XML.

Se um desenvolvedor já está acostumando a trabalhar com ferramentas XML ,como por exemplo DOM e XPath, então ele provavelmente também estará apto a trabalhar com SGBD XML nativo. O SGBD irá abstrair todos os detalhes de como XML é armazenada e deixar o desenvolvedor livre para construir aplicações utilizando tecnologias de XML.

3.2.2. Coleções

SGBD XML nativo gerenciam coleções de documentos, permitindo fazer consultas e manipulações com esses documentos como um conjunto. É muito parecido com o conceito relacional de tabela, sendo diferenciado por nem todos os SGBD XML nativo requererem um esquema para ser associado com a coleção. Isto significa que se pode armazenar qualquer documento XML na coleção, independente de esquema. Sendo assim, pode-se ainda construir consultas cruzadas em todos os documentos da coleção. SGBD XML nativo que dão suporte a esta funcionalidade são chamados independentes de esquema.

Com coleções independentes de esquema os SGBD possuem mais flexibilidade e facilitam o desenvolvimento de aplicações. Infelizmente, essa é uma característica que dá trabalho aos administradores de banco de dados devido ao risco de baixa integridade dos dados. Se for necessária uma estrutura de esquema mais rígida, deve-se ter certeza que o SGBD dá suporte a esquemas ou encontrar uma outra forma de armazenar o dado XML.

Alguns dos produtos no mercado dão suporte a validações com DTD e alguns podem confinar coleções inteiras de documentos via um esquema de linguagem proprietário. Em alguns anos é provável que o esquema de XML da W3C surgirá como esquema de linguagem escolhido para SGBD XML nativo. No entanto, os suportes atuais são limitados.

3.2,3. Consultas

XPath é a atual linguagem de consulta escolhida. No intuito de funcionar como as linguagens de consultas de SGBD, XPath é ligeiramente estendida para permitir consultas cruzadas de documentos. No entanto, ela não foi projetada para ser uma linguagem de consulta de SGBD.

Algumas das limitações de XPath inclui falta de agrupamento, junção de cruzamento de documentos e suporte a tipos de dados. Por causa desses assuntos, XPath precisa ser expandida para conter partes de uma linguagem mais completa. Algumas dessas limitações podem ser resolvidas utilizando XSLT (Extensible Stylesheet Language Family Transformations) [32], mas uma nova linguagem orientada a objetos está sendo desenvolvida no formato de XQuery. Alguns

fabricantes já começaram a liberar protótipos de implementações XQuery para serem usados em seus SGBD.

Para melhorar a performance das consultas, SGBD XML nativo dão suporte à criação de índices nos dados armazenados na coleção. Esses índices podem ser usados para melhorar a velocidade da execução das consultas. Os detalhes do que pode ser indexado e como os índices são criados variam entre produtos, mas a maioria dá suporte a esta característica de alguma forma.

3.2.4. Alterações

Alteração é a limiotação do atual SGBD XML nativo. A maioria dos produtos requer que o documento seja recuperado, alterado utilizando uma API XML, e então retornado ao SGBD. Alguns produtos possuem linguagens de alteração proprietárias e permitem alterar os documentos nos servidores. É provável que o problema de alteração seja resolvido quando for adicionada uma linguagem de alteração a XQuery. Até lá, manipulação com DOM é o método de alteração mais comum usado por SGBD XML nativo.

3.3. Arquitetura de SGBD XML Nativo

Existem dois tipos de arquitetura para bancos de dados XML nativos, uma baseada em textos e outra baseada em modelos.

3.3.1. Arquitetura Baseada em Texto

Um SGBD XML nativo com arquitetura baseada em texto é aquele que armazena XML como texto. Pode ser em um sistema de arquivos, um campo BLOB num SGBD relacional, ou ainda um formato de texto proprietário. Índices são comuns a todos os SGBD que apresentam esta arquitetura. Eles permitem que os mecanismos de consulta pulem facilmente para qualquer ponto no documento XML. Isso dá aos SGBD uma vantagem na velocidade quando recuperam documentos inteiros ou fragmentos de documentos. Isto porque o SGBD pode fazer apenas uma busca, posicionar uma única vez a cabeça de leitura/gravação no disco, e ,assumindo que o fragmento procurado está armazenado em bytes contínuos no disco, recupera o documento inteiro ou um fragmento numa única leitura. Por outro lado, remontando o documento a partir de pedaços, como é feito nos SGBD relacionais e orientados a objeto, requer múltiplas procuras de índices e várias leituras do disco.

Neste sentido, SGBD XML nativo baseados em texto são parecidos com os SGBD hierárquicos. Ambos podem atuar como um SGBD relacional quando recuperam e retornam dados

de acordo com uma hierarquia pré-definida. Ainda como SGBD hierárquicos, SGBD XML nativo com arquitetura baseada em texto estão aptos a encontrar problemas de performance na recuperação e no retorno de dados em qualquer outra forma, tal como invertendo a hierarquia ou pedaços dela.

3.3.2. Arquitetura Baseada em Modelo

Outra categoria de SGBD XML nativo são os que apresentam arquitetura baseada em modelo. Melhor do que armazenar documentos XML como textos, eles constroem um modelo de objeto interno a partir dos documentos e os armazena. Como o modelo é armazenado depende do SGBD. Alguns salvam o modelo em um SGBD relacional ou orientado a objetos. Por exemplo, armazenando DOM em um SGBD relacional pode resultar em tabelas como Elementos, Atributos, PCDATA, Entidades, por exemplo. OutrosSGBD utilizam um formato proprietário de armazenamento otimizado para seus modelos.

SGBD XML nativo baseados em modelos criados em outros SGBD podem apresentar uma performance similar a esses SGBD quando recuperam documentos, pelo motivo óbvio de confiar a recuperação de dados a estes sistemas. No entanto, o projeto do SGBD, especialmente para os XML nativos construídos sobre um SGBD relacional, apresentam grandes variações. Por outro lado, a maioria dos SGBD otimiza o modelo de armazenamento e também o software de recuperação.

SGBD XML nativo baseados em modelos que utilizam um formato proprietário de armazenamento costumam apresentar desempenho parecido com os que têm arquitetura baseada em texto quando recuperam dados na ordem em que foram armazenados. Isto ocorre uma vez que grande parte dos SGBD utiliza ponteiros físicos entre os nós, os quais apresentam performance similar à recuperação de textos.

Assim como os SGBD XML nativo baseados em texto, os com arquitetura baseada em modelo costumam ter problemas de desempenho quando precisam recuperar e retornar os dados em outra forma que não seja a que foi armazenada, assim como quando inverte a hierarquia ou partes dela. Não fica claro quando esta arquitetura é mais rápida ou mais lenta que a baseada em texto.

3.4. O Sistema Berkeley DB XML

Berkeley DB XML (BDB XML) [4] é um SGBD XML nativo com acesso a documentos armazenados e indexados por conteúdo. Sua linguagem de acesso é baseada em XQuery. Berkeley DB XML foi construído sobre o SGBD Berkeley DB (BDB) e herdou todas as suas

características e atributos. Assim como o BDB, o BDB XML é uma biblioteca, não um servidor, que apresenta uma API programável para desenvolvedores e roda em processo com a aplicação sem ajuda de uma administração humana.

Graças ao BDB e sua máquina de armazenamento subjacente, BDB XML herdou todas as propriedades ACID (atomicidade, consistência, isolamento e durabilidade). Ainda é possível armazenar ambos os formatos de dados, XML e não-XML, no SGBD Berkeley DB XML, o que pode ser uma vantagem para algumas aplicações. Segundo o fabricante, nenhum outro SGBD no mercado é baseado em uma tecnologia de campo e bem testada.

O SGBD Berkeley DB XML adicionou um *parser* de documento, uma indexação XML e uma máquina XQuery ao SGBD BerkeleyDB. Sua arquitetura pode ser verificada na Figura 1. XQuery está rapidamente se tornando a linguagem de consulta escolhida para acesso a dados XML. BDB XML dá suporte a XQuery 1.0 e XPath 2.0, e ainda a *namespaces* de XML, validação de esquema, operações nomeadas e cruzadas e *streaming* de documentos. A máquina XQuery usa um sofisticado otimizador de consulta baseado no custo e dá suporte a execuções de consultas pré-compiladas com variáveis embutidas. Documentos grandes podem ser armazenados intactos e quebrados em nós, permitindo recuperações mais eficientes e alterações parciais de documentos. BDB XML dá suporte ainda a uma indexação flexível dos nós XML, elementos, atributos e metadados para garantir uma recuperação do dado mais rápida e eficiente.

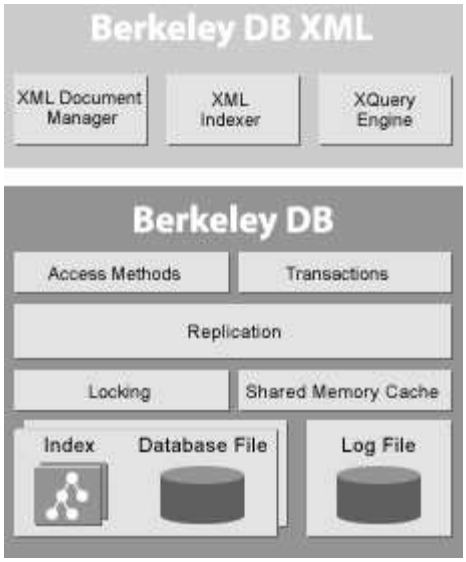


Figura 1 – Arquitetura do SGBD Berkeley DB XML

As principais características do SGBD Berkeley DB XML estão descritas nas subseções a seguir.

3.4.1. Armazenamento de Documentos XML

O Berkeley DB XML permite um armazenamento eficiente de documentos XML, indexação de dados flexível e rápido acesso utilizando XQuery ou Xpath. Como características do armazenamento de documentos pode-se destacar:

- Construído sobre o Berkeley BD, ele herdou todas as suas características, incluindo transações/recuperação ACID e replicação para uma alta disponibilidade;
- Armazenamento nativo e recuperação de dados XML e não-XML dentro de uma mesma transação, sem haver necessidade de um mapeamento ou uma translação;
- Controle de armazenamento flexível, por nós ou pelo documento inteiro;
- Agrupamento lógico de documentos;
- Validação de XML;
- Validação de esquema e de método por documento;
- Suporte a metadado de documento;
- Suporte a *namespace* de XML; e
- Preservação de espaços em branco através do armazenamento de todo o documento como uma unidade física.

3.4.2. Indexação de Documentos XML

Como o volume de dados XML cresce, um acesso eficiente torna-se crítico. O sistema único e dinâmico de indexação do Berkeley DB XML permite otimizar a recuperação do conteúdo XML. Sua máquina de consulta foi planejada focando estatísticas e custo, com o objetivo de entregar resultados rapidamente para consultas XQuery complexas. Como destaque temos:

- Indexação flexível dos nós, elementos, atributos e metadados;
- Criação de índices complexos e sua exclusão em tempo de execução;
- Índices focados em específicos *hot spot*;
- Tipo e existência específica de índices; e
- Planejamento interativo de consultas e otimização do índice.

3.4.3. Consulta de Acesso ao Documento XML

A linguagem XQuery traz aos bancos de dados XML o que o SQL traz aos bancos de dados relacionais. Com XQuery é fácil expressar relações complexas, agrupamentos, condições e conjuntos de resultados em consultas que podem ser otimizadas e executadas de forma rápida sob um conjunto enorme de dados. O Berkeley DB XML se aproxima aos padrões do XQuery e oferece uma das mais completas implementações. Como características principais pode-se destacar:

- Suporte a XQuery versão 1.0 e XPath 2.0;
- Consultas em uma única tabela ou com cruzamento de várias tabelas;
- Identificadores permanentes aos documentos para acesso direto;
- Otimização da consulta através da máquina de consulta baseada no custo;
- Consultas pré-compiladas contendo variáveis para uma execução repetida ainda mais eficiente;
- *Streaming* de documento a partir de arquivo ou memória; e
- Navegação no conjunto de resultados como no DOM.

3.4.3. Modificações no Documento XML

O banco de dados Berkeley DB XML fornece uma API completa para modificações permitindo alterações muito eficientes. Alterações no documento XML ainda não fazem parte do padrão XQuery, no entanto assim que estes padrões forem aceitos, BDB XML dará suporte a estas modificações. Como características temos:

- Alteração parcial de documentos;
- Modificação de documentos em pontos específicos com transações; e
- Modificação concorrente em diferentes seções do conteúdo.

3.4.5. Distribuição

O Berkeley BD XML é muito flexível, fácil de distribuir e fácil de integrar. Assim como um conjunto de bibliotecas C e C++, ele pode ser instalado e configurado em alguma aplicação. Ele pode ser integrado com o Apache e pode ser acessado diretamente usando código em Java, C++ ou a partir de página na Web utilizando PHP, tornando-o ideal para sistemas Web dinâmicos. BDB XML foi projetado para operar de uma forma completamente desacompanhada, então todas as funções administrativas são controladas programavelmente. Ele dá suporte a uma grande

variedade de linguagens de programação e plataformas de sistemas operacionais. Quanto à distribuição suas características são:

- Gerenciamento e administração programadas, não há necessidade de nenhuma administração humana;
- Ferramentas de linhas de comando para carregar, fazer backup, e interação com bases de dados XML;
- Suporte a várias linguagens (C++, Java, Perl, Python, PHP, Tcl, Ruby);
- Suporte a vários sistemas operacionais (Windows, Linux, BSD Unix, Mac OS/X);
- Instalador para Microsoft Windows; e
- Integração com Apache.

3.5. SGBD Oracle 10g

Desde sua introdução no Oracle 9i Release 2, o XML DB do Oracle recebeu revisões profundas no sentido de esconder a complexidade de gerenciar dado XML, oferecendo um rico conjunto de características de consulta e entregando funcionalidades adicionais como a evolução do esquema e WebDav [15,16]. Como resultado, a adoção do XML DB se intensificou entre as empresas se tornando líder no gerenciamento de dados e conteúdo de forma única.

Construído sobre o SGBD Oracle, o Oracle XML DB oferece um armazenamento de forma eficiente, recuperação, consulta, geração, e gerenciamento de grandes volumes de dados XML. Desde sua inserção, o Oracle XML DB forneceu uma profunda integração de XMLType, processamento de XML Schema, armazenamento estruturado e não-estruturado, repositório de conteúdo, SQL/XML e capacidade de gerenciamento. Com cada nova versão do SGBD Oracle, o XML DB também continuou a envolver mais versatilidade, escalabilidade e características eficientes em várias áreas-chave. Na nova versão do Oracle Database 10g Release 2, o Oracle XML DB introduz um poderoso esquema de indexação XMLIndex, uma implementação eficiente de padrões baseados em XQuery, um versátil esquema baseado no recurso de metadados, um conjunto de novas funções para operações DML no dado XML e várias outras funcionalidades.

3.5.1. Arquitetura

Como mostra a Figura 2, o Oracle XML DB tem dois componentes principais:

- Armazenamento das tabelas e visões XMLType; e
- Repositório Oracle XML DB.

Existe ainda um rico conjunto de serviços associados, protocolos e API para esses dois componentes para dar suporte a numerosos cenários.

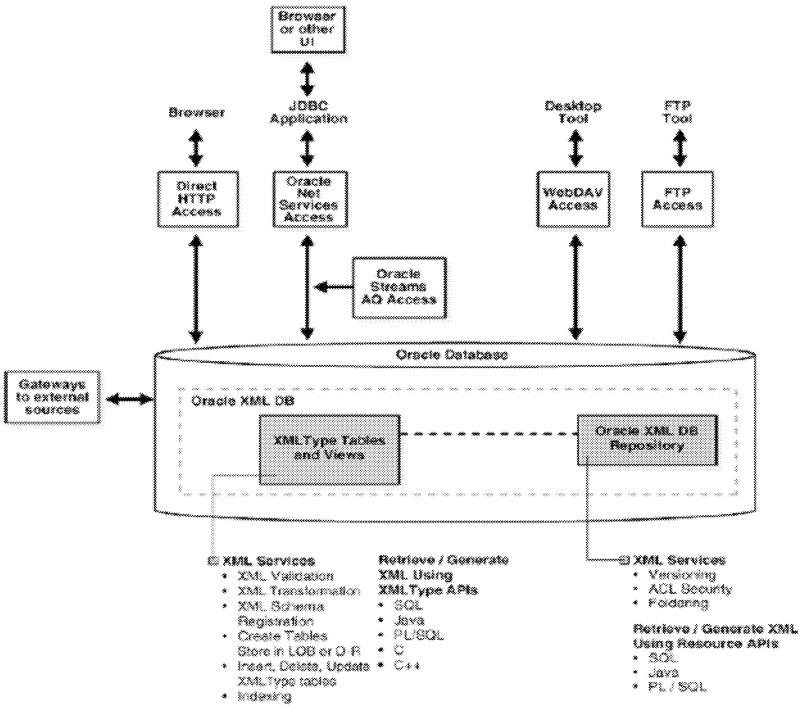


Figura 2 - Oracle XML DB.

3.5.2. Armazenamento XMLType

O armazenamento XMLType no Oracle XML DB tem um conjunto de componentes essenciais como pode ser verificado na Figura 3. Quando os esquemas XML são registrados com o Oracle XML DB, um conjunto de tabelas default são criadas e usadas para armazenar instancias de documentos XML associados com o esquema. Estes documentos também podem ser vistos e acessados no Repositório Oracle XML DB.

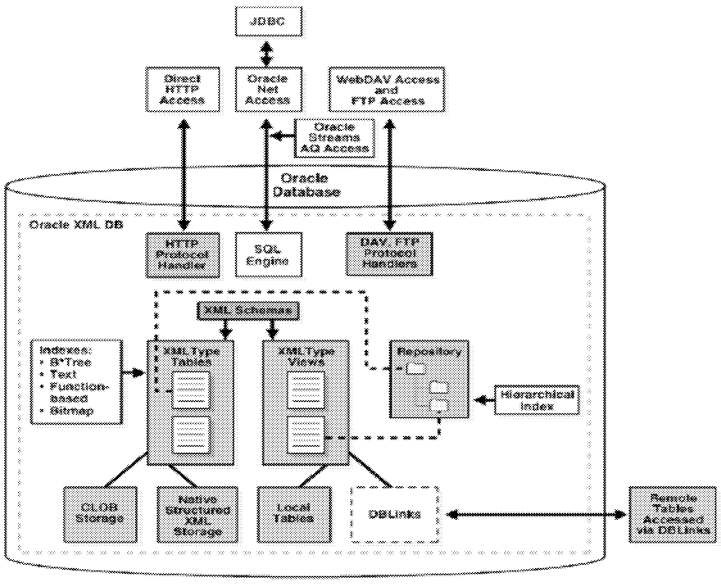


Figura 3 – Componentes do armazenamento XMLType no Oracle XML DB

Tabelas e colunas XMLType podem ser armazenadas como valores Character Large Object (CLOB) ou como um conjunto de objetos. Quando armazenados como um conjunto de objetos, quer-se dar referência à estrutura.

Os dados nas visões XMLType podem ser armazenados localmente ou em tabelas remotas. Tabelas remotas podem ser acessadas via ligações.

Ambas, tabelas e visões XMLType, podem ser indexadas por árvores B, XMLIndex, Oracle Text, índices baseados em funções ou índices bitmap.

Podemos acessar dados no Repositório Oracle XML DB utilizando qualquer um dos seguintes modos:

- HTTP e HTTPS concorrentes, pelo manipulador de protocolo HTTP;
- WebDAV e FTP, pelo servidor de protocolo FTP e WebDAV; e
- SQL, pela rede de serviços Oracle, incluindo JDBC.

Oracle XML DB dá suporte a mensagens de dados XML utilizando o Oracle Streams Advanced Queuing (AQ) e Web Services.

3.5.3. Repositório Oracle XML DB

O Repositório Oracle XML DB é um componente do SGBD Oracle que otimiza o manejo de dados XML. Este componente contém recursos que podem ser ou pastas (diretórios) ou arquivos. Cada recurso é identificado por um caminho. Um recurso também tem um conjunto de metadados definidos pelo sistema, além de seu conteúdo. Ele pode ainda conter metadados definidos pelo usuário, informação esta que não é parte do conteúdo e sim associadas a ele.

Apesar de o Repositório Oracle XML DB manipular conteúdo XML em particular, podemos utilizá-lo para armazenar outros tipos de dados além de XML. De fato, podemos utilizar o repositório para acessar qualquer tipo de dado armazenado no SGBD Oracle.

Podemos acessar dados no Repositório Oracle XML DB das seguintes maneiras:

- Utilizando SQL, pelas visões RESOURCE_VIEW e PATH_VIEW;
- Usando PL/SQL, pela API DBXML_XDB; e
- Utilizando Java, pelo recurso da API Oracle XML DB para Java.

Além de fornecer API que acessam e manipulam dados, o Repositório Oracle XML DB fornece API para os seguintes serviços:

- Versão. - O Oracle XML DB usa o pacote DBMS_XDB_VERSION PL/SQL para criar versões dos recursos no Repositório Oracle XML DB. Alterações subseqüentes ao recurso criam novas versões;
- Segurança ACL - O recurso de segurança do Oracle XML DB é baseado no Acesso Controlado de Listas (ACL). Todo recurso tem um ACL associado que lista seus privilégios. Todas as vezes que o recurso é acessado ou manipulado, o ACL determina se a operação é permitida; e
- Pastas - O Oracle gerencia uma hierarquia de recursos de pastas que contém outros recursos.

O dado em uso normalmente está associado a informações adicionais que não fazem parte do conteúdo. Podemos utilizar esse metadado para agrupar ou classificar dados e processá-los de formas diferentes. Por exemplo, em uma coleção de fotografias digitais podemos querer associar metadados à figura, como localização ou pessoas. No Repositório Oracle XLM DB podemos associar recursos com o metadado definido. Cada repositório possui ainda metadados

gerados pelo Oracle associados a ele. Estes metadados são criados automaticamente e usados de forma transparente para gerenciar o recurso.

Na Figura 4 podemos observar a arquitetura do Repositório Oracle XML DB. Um recurso é qualquer pedaço de conteúdo gerenciado pelo Oracle. Cada recurso possui um nome, uma lista de controle de acesso associada que determina quem pode ver o recurso, algumas propriedades estatísticas e outras propriedades extensíveis pela aplicação.

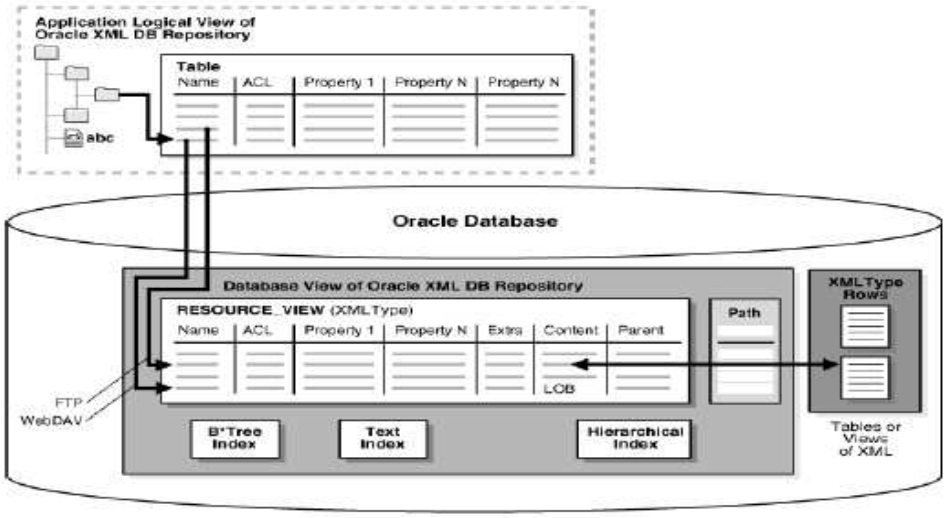


Figura 4 - Repositório Oracle XML DB

O modelo relacional tabela-linha-coluna é aceito como um mecanismo eficiente para gerenciar estrutura de dados. No entanto, o modelo não é tão efetivo quando se trata de dados semi-estruturados e não estruturados, como documentos XML. Como exemplo, temos o armazenamento de um livro: não é tão simples representá-lo em linhas seria muito mais fácil retratá-los hierarquicamente, *livro:capítulo:seção:parágrafo*. São características do Repositório Oracle XML DB:

- Gerenciamento hierárquico de documentos centrados em conteúdo XML;
- Índices hierárquicos que acessam pastas e caminhos transversais; e
- Pode-se acessar documentos XML utilizando protocolos como FTP, HTTP, HTTPS e WebDAV e ainda por linguagens como SQL, PL/SQL, Java e C.

SGBD relacionais são tradicionalmente pobres quando se fala em armazenamento de estruturas hierárquicas. O Oracle XML DB fornece um repositório organizado hierarquicamente que pode ser consultado e permite o gerenciamento do conteúdo de um documento XML.

3.5.4. API para Oracle XML DB

Todas as funcionalidades do Oracle XML DB são acessíveis pelas API JDBC, PL/SQL, OCI e ODP.NET. Os métodos mais comuns para construirmos aplicações Web são os baseados na arquitetura .NET ou J2EE. Com a arquitetura J2EE, JDBC pode ser utilizado para acessar o XML DB. Já no ambiente .NET, a API do Oracle ODP.NET permite o acesso ao XML DB utilizando as linguagens Visual Basic, C# ou C/C++.

3.5.5. Características do Oracle XML DB

Todo SGBD construído para gerenciar XML deve ser capaz de armazenar documentos XML. O Oracle XML DB é capaz de muito mais do que isso. Ele fornece características padrões de SGBD como controle de transação, integridade dos dados, replicação, segurança, escalabilidade, confiabilidade e disponibilidade ao mesmo tempo que procura eficiência na indexação, nas consultas, nas alterações e na pesquisa a documento XML.

3.5.6. XMLType

XMLType é um tipo de dados nativo ao servidor que permite ao SGBD perceber que uma coluna ou uma tabela contém dados XML. É análogo ao tipo de dados DATE, o qual permite ao SGBD perceber que uma determinada coluna contém uma data. XMLType fornece ainda métodos que permitem operações comuns como validação do XML Schema. Utilizamos o tipo de dado XMLType como qualquer outro tipo de dado, criando colunas e definindo variáveis em PL/SQL. Como XMLType é um tipo objeto também podemos criar tabelas do tipo XMLType. Por default, uma tabela ou coluna deste tipo pode conter qualquer documento XML bem formado.

O Oracle XML DB armazena o conteúdo do documento como texto XML utilizando o tipo de dado Character Large Object(CLOB). Isto permite uma maior flexibilidade em termos da forma das estruturas de XML que podem ser armazenadas em uma única tabela ou em coluna.

3.5.7. XML Schema

Com suporte pela W3C, o XML Schema é uma característica chave no Oracle XML DB. O XML Schema especifica a estrutura, o conteúdo e certas semânticas de um conjunto de documentos XML.

O XML Schema unifica o documento e o modelo de dados. No Oracle XML DB, podemos criar tabelas e tipos automaticamente utilizando XML Schema. Em resumo, isto significa que podemos desenvolver e usar padrões de modelos de dados para os dados, sejam eles

estruturados, não estruturados ou semi-estruturados. O Oracle dá suporte a todas as construções definidas pela Recomendação XML Schema, exceto a construção de redefinição. Ele ainda inclui métodos e funções SQL que permitem ao esquema XML validar uma instância de um documento conforme sua especificação.

3.5.8. Armazenamento Estruturado X Armazenamento Não Estruturado

Uma decisão que deve ser tomada quando formos usar o Oracle XML DB é quanto ao armazenamento do documento XML, se deve ser estruturado ou não estruturado. Para melhorar a performance devemos escolher também o correto esquema de indexação para ambas as opções de armazenamento.

No armazenamento não-estruturado, todo o documento XML é armazenado inteiro em um CLOB. Este tipo de armazenamento fornece o maior *throughput* possível na inserção e na recuperação de documentos XML. No entanto, pouco pode ser feito para otimizar consultas e alterações nos documentos quando se utiliza este tipo de abordagem.

O armazenamento estruturado apresenta grande vantagem no gerenciamento de documentos XML, incluindo um gerenciamento otimizado da memória, redução nas exigências de armazenamento, indexação por árvores B, e alterações localizadas (*in-place*). Este tipo de armazenamento requer um processamento maior do esquema XML correspondente na hora de inserir e recuperar o documento inteiro.

3.5.9. Duplicidade XML/SQL

Um objetivo chave do Oracle XML BD é fornecer duplicidade XML/SQL. Isto significa que o programador XML pode usufruir do poder de um modelo relacional quando está trabalhando com conteúdo XML, enquanto um programador SQL pode usufruir da flexibilidade de XML quando está trabalhando com conteúdo relacional. Isto fornece o desenvolvimento de aplicações com maior flexibilidade, permitindo-as utilizar as ferramentas mais apropriadas para um problema em particular.

Duplicidade XML/SQL significa que um mesmo dado pode ser exposto em linhas como em uma tabela e manipulado usando SQL, ou exposto como nós em documentos XML e manipulados utilizando técnicas como DOM e transformações XSL. O acesso e o processamento são totalmente independentes do formato do armazenamento.

Desde o Oracle 9i Release 2, características XML/SQL têm suporte como parte integral do XML DB. Este também inclui um número de extensões adicionais XPath baseado em SQL para

apoiar consultas, alterações e transformações de dados XML. Com as funções XML/SQL podemos acessar conteúdo XML em qualquer parte de uma consulta SQL. As operações SQL/XML mais importantes são:

- existsNode – usado com a cláusula WHERE de uma consulta SQL com o objetivo de restringir o conjunto de documentos retornados por uma consulta;
- extract – esta função pega uma expressão XPath e retorna os nós que casam com a expressão, como um documento XML ou um fragmento;
- extractValue – esta função pega uma expressão XPath e retorna o nó folha correspondente;
- insertChildXML – insere um nó, elemento XML ou um atributo, como filho de um determinado nó pai;
- insertXMLbefore – insere um nó XML de qualquer tipo imediatamente antes de um determinado nó (menos o nó atributo);
- appendChildXML – insere um nó XML de qualquer tipo como o último nó filho de um determinado nó;
- deleteXML – remove um nó XML de qualquer tipo;
- updateXML – permite alterações parciais em documentos XML, baseado em um conjunto de expressões Xpath; e
- XMLSequence – permite expor os membros de uma colação como uma tabela virtual.

3.5.10. XQuery Nativo do Oracle

XQuery 1.0 é uma linguagem de consulta XML desenvolvida pela W3C que será recomendada como linguagem de consulta a documentos XML. Várias empresas estão adotando XQuery como a forma de consultar os documentos XML armazenados nas linhas dos bancos de dados ou por WebServices.

Pelo lado de SQL, o tipo de dado XML foi introduzido no SQL 2003 [18] como um modo de encapsular XML em SQL. O comitê do SQL está agora trabalhando para integrar a consulta a XML utilizando XQuery. Isto está sendo realizando pela introdução de uma nova função SQL, XMLQuery, e um novo construtor, XMLTable. Ambos operam os valores XML e SQL utilizando XQuery. O formato é conhecido como XQuery-centric e permite consultas e construções de dados XML utilizando XQuery.

O SGBD Oracle 10g Release 2 dá suporte a XQuery no servidor de banco de dados utilizando os padrões de funções SQL.

Os benefícios do suporte nativo a XML do Oracle 10g estão listados nas subseções a seguir.

3.5.12. Unificação de dados e conteúdo

A maioria dos dados das aplicações e dos conteúdos da Web é armazenada em SGBD relacionais ou em um sistema de arquivos. XML é normalmente utilizado como meio de transporte e é gerado a partir de um SGBD ou de um arquivo. Isto porque o volume de dados XML aumenta o custo para gerá-lo novamente e os métodos de armazenamento se tornam menos efetivos em acomodar conteúdo XML.

Com o Oracle XML DB, podemos armazenar e gerenciar dados que estão estruturados, não-estruturados e semi-estruturados utilizando um padrão de modelo de dados e padrões SQL e XML. Podemos realizar operação SQL em documentos XML e operações XML em dados objeto-relacional (como tabelas).

3.5.13. Armazenamento e Recuperação mais Rápida de Documentos XML Complexos

Os usuários hoje em dia se deparam com uma barreira de desempenho em armazenamentos e recuperações complexas, grandes, ou de muitos documentos XML. O Oracle XML DB fornece um alto desempenho e escalabilidade nas operações XML. As principais características do desempenho são:

- XMLType nativo;
- Integração de XPath e suporte a XSLT;
- Suporte a XML Schema; e
- Índice hierárquico sobre o Repositório Oracle XML DB.

3.5.14. Fácil Integração de Aplicação

O Oracle XML DB permite que os dados de sistemas remotos sejam acessados por *gateways* e combinados em um modelo de dados em comum. Isto reduz a complexidade no desenvolvimento de aplicação que trabalha com dados de diferentes bases.

4. Representação de Mídias e Objetos de Realidade Virtual

Este trabalho tem como objetivo fazer o gerenciamento de mídias e de objetos de realidade virtual desde seu armazenamento até sua exibição no browser. Para isso é necessário termos conhecimento de como estes objetos são representados.

A linguagem VRML [24] ganhou destaque ao longo dos anos na modelagem de ambiente e objetos virtuais por permitir um desenvolvimento mais detalhado e iterativo com a Web.

Nos últimos anos, o desenvolvimento da linguagem X3D [25] corrigiu vários problemas relacionados com as tradicionais linguagens de modelagem de objetos 3D. X3D é uma extensão de VRML no formato XML, possibilitando a utilização de tecnologias relacionadas a XML para manipular os mundos em X3D.

4.1. VRML

Virtual Reality Modeling Language (VRML) [24,2] está para a realidade virtual assim como o HTML está para textos. É um formato independente de plataforma, estruturado e padronizado para conteúdos estáticos ou iterativos. Assim como em HTML, VRML pode ser escrita à mão ou gerada por um programa. Esta ultima é a opção mais usada.

Alguns anos atrás VRML estava ganhando velocidade, mas muito do entusiasmo parece ter desvanecido. Ela não dominou o mercado como uma tecnologia básica para o desenvolvimento na Web juntamente com HTML. Uma razão para isso é que não existe no mercado uma versão de um browser VRML que funcione para Linux e outros sistemas operacionais UNIX. Os que existem são incompletos e geralmente não conseguem exibir o conteúdo do VRML de forma apropriada. Por exemplo, eles não dão suporte para lidar com nós Script, que é uma das principais alterações do VRML 97. Desta forma, é muito provável que o grau de aceitação de VRML esteja ligado a uma enorme fração de usuários que utilizam Linux e não querem dar apoio a uma aplicação que não funciona ou funciona de forma precária.

Contudo, VRML apresenta algumas vantagens em relação às demais linguagens 3D. VRML dá suporte a muitas ordenações da interação na cena e ainda ao envio de diagramas por e-mail, por exemplo.

4.1.1. Conceitos Básicos

Para dar início ao estudo de conceitos básicos, vamos tomar como exemplo a Figura 5. A figura em questão mostra uma esfera de raio 0,5 (normalmente em VRML chama-se as unidades

de metros, no entanto isto é apenas uma convenção). Esta esfera é a “geometria” de um nó “formato”, cuja “aparência” tem a “cor difusa” de 0.8 0 0, ou seja, vermelho bem brilhante. Consequentemente, a forma realmente descreve uma esfera vermelha. Esta forma é um dos “filhos” de uma “transformação”, cuja translação é 0 1 0, isto significa que está a um metro acima em relação ao eixo Y. O Quadro 6 exibe o código responsável pela geração da esfera. A primeira linha é um comentário que avisa ao browser que o arquivo foi escrito na versão 2.0 de VRML.

Figura 5 – Esfera Vermelho Brilhante

Quadro 6 – Código VRML para a Esfera da Figura 5

```
#VRML V2.0 utf8
   Transform {
   translation 0 1 0
   children [
       Shape {
         appearance Appearance {
            material Material {
                diffuseColor 0.8 0 0
            }
         }
         geometry Sphere {
            radius 0.5
         }
      }
   ]
}
```

Em resumo, o exemplo anterior descreve uma esfera com um raio de meio metro situada a um metro acima da origem. O código do Quadro 6 parece um jeito de descrever uma cena tão simples de forma grosseira. O código do Quadro 7, a seguir, parece ser mais simples:

Quadro 7 – Tentativa de Simplificação do Código do Quadro 6

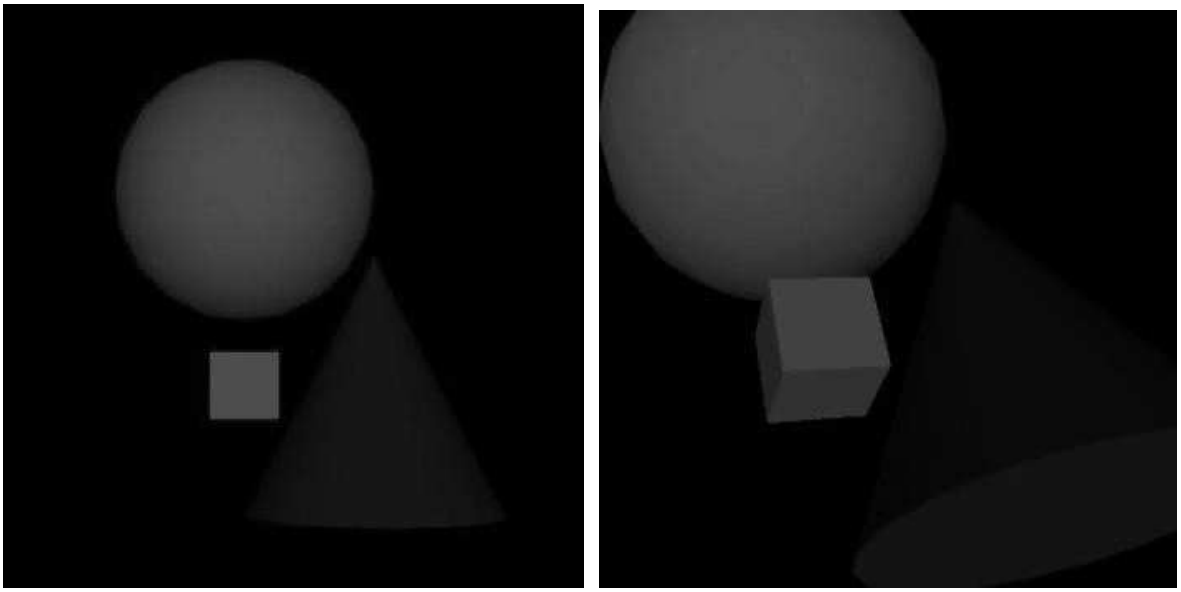
```
# Este codigo nao eh valido em VRML
Sphere {
        center 0 1 0
        radius 0.5
        color 0.8 0 0
}
```

De fato, é possível criar nós apropriados para aplicações próprias, utilizando protótipos. O código do Quadro 8 mostra um exemplo.

Quadro 8 – Uso de Protótipos

```
#VRML V2.0 utf8
# Define um Prototipo:
# A Thing with a given center and color
PROTO Thing [
   field SFVec3f center 0 0 0
   field SFColor color 0.8 0 0
   field SFNode thing NULL
] {
   Transform {
      translation IS center
      children [
         Shape {
            appearance Appearance {
               material Material {
                  diffuseColor IS color
               }
            }
            geometry IS thing
         }
      ]
   }
}
# Use the PROTOtype for displaying various
things
Thing {   # Small red box at origin
   thing Box { size 0.5 0.5 0.5 }
}
Thing {   # Red sphere at 1.5 meters up from
origin
   center 0 1.5 0
   thing Sphere { }
}
Thing {   # Blue cone at 1 meter right from
origin
   center 1 0 0
   color 0 0 0.8
   thing Cone { }
}
```

Os objetos criados pelo código do Quadro 8 são exibidos nas Figura 6 (a) e (b) em dois ângulos diferentes. Assim, é possível definir qualquer quantidade de cenas como as da Figura 6, utilizando o protótipo que foi definido no Quadro 8. Obviamente é necessário incluir o protótipo original do começo. Ainda é possível referenciá-los em outro arquivo utilizando protótipos externos, os quais incluem parte de uma interface mas omitem sua definição. Se isto parecer muito esforço de digitação, é possível ainda criar um script em outra linguagem de programação, exemplo Java, para incluir o protótipo correto nos arquivos VRML, ou para fazer qualquer outro trabalho repetitivo.



(a) (b)

Figura 6 – Objetos Gerados pelo Código do Quadro 6 em ângulos Diferentes

Nos grafos da cena, pode-se dar nome a um nó usando a indicação DEF e utilizá-lo novamente em outro lugar usando a indicação USE. Vejamos o exemplo do Quadro 9.

Quadro 9 – Exemplo das Indicações DEF e USE

```
#VRML V2.0 utf8
Shape {
   appearance DEF APP Appearance {
      material Material {
         diffuseColor 0.8 0 0
      }
   }
   geometry Box { }
}
Transform {
   translation 0 2 0
   children [
       Shape {
         appearance USE APP
         geometry Sphere {
            radius 0.5
         }
      }
   ]
}
```

Note que diferentemente do uso de protótipos, onde é feita uma cópia completa do conteúdo do protótipo, somente um nó “aparência” está presente, o qual é referenciado em dois lugares. Se for preciso fazer uma animação posteriormente para mudar a cor da esfera de vermelho para azul, por exemplo, ambos, o quadrado e a esfera, mudarão de cor, enquanto que a animação da cor de “Thing” no Quadro 8 somente ele especificamente será animado. Uma vez que um nó for nomeado DEF, ele pode ser utilizado com indicações USE qualquer quantidade de vezes.

Reafirmando tudo o que já foi discutido neste trabalho sobre VRML, podemos dizer que o mundo VRML é descrito por uma hierarquia de nós chamadas grafos de cenas. Cada nó é um tipo particular de nó, e cada tipo de nó da especificação VRML97 define vários campos. Cada campo tem um tipo e um valor default. A sintaxe para um nó é mostrada no Quadro 10.

Quadro 10 – Sintaxe de Nó

```
NodeType {
        field value
        field value
        ...
}
```

A sintaxe para um valor depende do tipo do campo. O valor de um SFVec3f, por exemplo, é formado por três números com ponto flutuante. Já o valor de um SFNode é outro nó como o exibido acima.

Os tipos dos campos com valores únicos são:

- SFBool - um valor booleano, escrito TRUE ou FALSE;
- SFFloat - um pontoflutuante;
- SFImage - uma imagem, descrita por vários valores inteiros, primeiro largura, depois a altura e o número de componentes, finalmente o número de pixels (altura x largura);
- SFInt32 - um inteiro 32-bit;
- SFNode - um nó;
- SFRotation - uma rotação, três valores de pontos flutuantes para o eixo e um para o ângulo em radianos;
- SFString - uma string entre aspas duplas. Dentro da string, aspas duplas e a barra invertida são precedidas de barra invertida;
- SFTime - um valor de ponto flutuante, tempo em segundos;
- SFVec2f - um vetor bi-dimencional contendo dois valores de pontos flutuantes; e
- SFVec3f - um vetor tri-dimencional contendo três pontos flutuantes.

Para a maioria dos tipos de campo com valores únicos existe um correspondente tipo de campo com múltiplos valores, o qual significa um ou mais valores do correspondente tipo de valor único. Como exemplo, o código do Quadro 11 apresenta valores legais para o campo MFVec3f:

Quadro 11 – Valores Válidos para o Campo MFVec3f

```
[]
0.1 2 3
[0.1 4 2]
[0.5 2 6 1 6 4 7 4 6]
[0.5 2 6, 1 6 4, 7 4 6]
```

Isto é, os valores podem ou não ser separados por vírgulas e devem estar entre colchetes se existir mais ou menos valores de um dos campos de valores simples.

Os nomes dos tipos dos campos também são usados dentro da declaração do protótipo. A sintaxe do protótipo é, basicamente, a mostrada no Quadro 12.

Quadro 12 – Sintaxe do Protótipo

```
PROTO Name [
        field Type fieldName defaultvalue
        exposedField Type fieldName
defaultvalue
        eventOut Type fieldName
        eventIn Type fieldName
        ...
] {
        Node { ... }
        ...
}
```

Dentro do corpo da definição dos protótipos, os valores dos campos podem também ser especificados pela indicação IS.

4.1.2. Animação e Interação

Criar cenas estáticas não é muito complicado. Mas a parte interessante de VRML é a habilidade de interagir com o usuário. Na seção anterior foi mostrado como um nó descreve a cena a ser renderizada. O que não foi mencionado foi o fato dos nós poderem enviar eventos uns para os outros em rotas especificadas. As rotas de evento são independentes da hierarquia da cena.

Rotas são especificadas pelas indicações ROUTE. O exemplo do Quadro 13 cria um cubo branco (Figura 7), o qual muda lentamente para vermelho e para azul quando houver o clique do *mouse* sobre ele, e depois retorna para branco em 2,5 segundos depois do clique do *mouse*. O que acontece quando há o clique do *mouse* é que o nó “botão” sente o clique e manda o evento “touchTime”. A indicação ROUTE dispara o evento que deve ser roteado no “startTime” do “TimeSensor” TS, o qual dispara a geração de eventos de tempo para um ciclo (o ciclo tem duração de 2,5 segundos). Para cada tique-taque do relógio, o “TimeSensor” envia um evento chamado fraction_changed com um valor SFFloat entre 0 e 1 (dando a fração do tempo que já decorreu). Este evento é então roteado para o ColorInterpolator CI, ou seja, ele faz uma interpolação linear dos valores das cores. O CI então manda o evento value_changed para o “Material” MAT que o recebe como uma “cor difusa”, fazendo a cor do cubo mudar. Se o CI receber um evento set_fraction com valor 0,3, por exemplo, ele checaverifica se é um campo chave e verifica que 0,3 está entre o primeiro e o segundo valor. Ele é seis décimos do caminho para o segundo valor, então a saída é a cor (0.6 0.6 0.6).

Quadro 13 – Código de Criação do Cubo Mostrado na Figura 8

```
#VRML V2.0 utf8

DEF TS TimeSensor {
 cycleInterval 2.5
}

DEF CI ColorInterpolator {
   key [0 0.33 0.67 1]
   keyValue [1 1 1, 1 0 0, 0 0 1, 1 1 1]
}

Group {
 children [
 DEF BUTTON TouchSensor { }
 Shape {
 geometry Box { }
 appearance Appearance { material DEF MAT Material { } }
 }
 ]
}

ROUTE BUTTON.touchTime TO TS.startTime
ROUTE TS.fraction_changed TO CI.set_fraction
ROUTE CI.value_changed TO MAT.diffuseColor
```

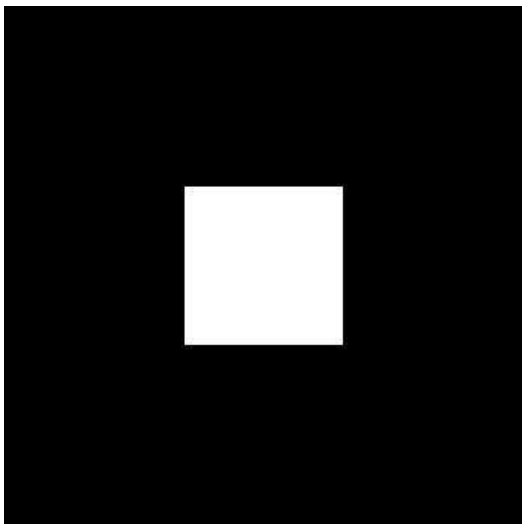


Figura 7 – Cubo Descrito no Quadro 13

O fato dos eventos utilizarem um interpolador para alcançar o destino é muito comum em VRML. Tudo isto permite especificar mapeamentos arbitrários relativamente fáceis e baratos.

Alguns tipos diferentes de interpoladores estão na especificação VRML97, para diferentes tipos de dados.

4.1.3. Scripting

A desvantagem dos interpoladores é que eles não conseguem focar um ponto particular em uma seqüência. Seria mais fácil para os autores da especificação VRML97 adicionar um tipo de nó que possibilitasse isto, porém os desenvolvedores da linguagem escolheram não fazê-lo. Ao invés disso, eles escolheram criar uma interface externa geral para linguagens de script, Java e JavaScript. Podemos definir comportamentos arbitrários para o “mundo” usando estas linguagens de programação.

No exemplo anterior do Quadro 13, para que o cubo desapareça e seja substituído por uma pequena esfera azul nos primeiros 1,5 segundos do ciclo, por exemplo, é preciso utilizar nós de scripts. O código deste exemplo em particular é exibido no Quadro 14. Os nós TS e CI e as rotas entre eles e o MAT são as mesmas do exemplo anterior.

Quadro 14 – Alterando o Código do Quadro 13

```
#VRML V2.0 utf8

DEF SCR Script {
      eventIn SFFloat set_fraction
      field SFInt32 curchoice 0
      eventOut SFInt32 choice
      url "javascript:
         function set_fraction(val,time) {
            if(val < 0.6 && curchoice == 0) {
               choice = 1;
               curchoice = 1;
         } else if(val >= 0.6 && curchoice == 1) {
               choice = 0;
               curchoice = 0;
            }
         }
      "
   }
DEF BUTTON TouchSensor { }
   DEF SW Switch {
    choice [
      Shape {
       geometry Box { }
       appearance Appearance { material DEF MAT
          Material { } }
      }
      Shape {
       geometry Sphere { radius 0.5 }
       appearance Appearance { material Material
          { diffuseColor 0 0 0.8 } }
```

```
    }
  ]
}
}
ROUTE TS.fraction_changed TO SCR.set_fraction
ROUTE SCR.choice TO SW.whichChoice
```

Como pode ser observado no exemplo, a sintaxe para especificar campos internos aos nós de Script é similar àseção da interface do protótipo. Apenas, no lugar de dizermos fieldName value dizemos kind Type fieldName seguido pelos valores dos campos. Isto ocorre porque a especificação define somente três campos para os nós Script: url, directOutput e mustEvaluate, e deixa ao programador a definição dos eventos de entrada, os campos e os eventos de saída do seu script.

No exemplo do Quadro 14, o script interno ao nó Script é especificado no campo url. Neste caso, o código é escrito em JavaScript e encaixado na URL. Também seria possível escrever o script em um arquivo (com a terminação .js) e referenciá-lo na URL. Este script poderia ainda ser escrito em outra linguagem como, por exemplo, Java.

4.1.4 Aplicações

Até então foi discutido como é a escrita de VRML. No entanto, os resultados mais interessantes vêem da criação automática de VRML. Mundos estáticos, uma vez programados, são interessantes especialmente para jogos, arte e propagandas. Ótimas possibilidades continuam a ser exploradas na interface entre as informações do mundo e o *browser* 3D.

É possível usar uma API fornecida pelo browser para criar aplicações que usam VRML para fornecer uma parte da Graphical user Interface (GUI). Podemos ainda acessar o browser VRML por uma API de Java chamada External Authoring Interface (EAI). Isto possibilita ao programador escrever *applets* na web que acessam a cena VRML.

4.2. X3D

EXtensible 3D Graphics (X3D) [25] é um padrão aberto da ISO para distribuir conteúdo 3D. Ele não é uma API de programação, tampouco é somente um formato de arquivo para intercâmbio de geometrias. Ele combina ambos, geometria e descrição do comportamento em tempo de execução, em um único arquivo que possui um número de diferentes formatos de arquivos disponível, incluindo XML. X3D é a nova revisão da especificação ISO de VRML97, incorporando os últimos avanços das características doque utilizam VRML97.

A tecnologia X3D foi escolhida para representar objetos virtuais e mídias no módulo de gerenciamento proposto por este trabalho, pela facilidade de ser descrita no formato XML.

4.2.1. Diferença entre o VRML e X3D

Existem grandes e pequenas diferenças entre os padrões VRML e X3D. Em principio, X3D parte de VRML e esclarece todas as áreas obscuras da especificação VRML97 que não foram cobertas em anos de uso. Assim, partiu da premissa básica e estendeu-a fornecendo uma maior flexibilidade. As principais mudanças incluíram a finalização da especificação, dividindo-a em três especificações separadas, lidando com conceitos abstratos, codificação do formato dos arquivos e acesso às linguagens de programação. Ainda são mais precisos com os modelos de evento e de "*lighting*", e modificaram alguns nomes de campos para uma melhor consistência.

As principais mudanças podem ser resumidas nos pontos a seguir:

- Aumento da capacidade de cenas gráficas;
- Revisão e unificação do modelo de programação de aplicações;
- Vários arquivos de codificação descrevendo o mesmo modelo de abstração incluindo XML;
- Arquitetura modular permitindo uma escala de níveis de adoção e suporte para vários tipos diferentes de mercado; e
- Estrutura da especificação expandida.

O elemento principal de uma aplicação X3D é o grafo de cena X3D. O grafo de X3D é muito parecido com o da especificação VRML97 que teve como base de construção uma técnica de computação gráfica interativa e já conhecida. Durante anos este grafo atendeu bem às necessidades de VRML e resistiu ao tempo. O grafo de cena de X3D sofreu alterações já no seu projeto inicial para atender aos avanços do mercado de dispositivos gráficos, acrescentando-se novos nós e campos de dados. Outras pequenas alterações tiveram destaque de modo a esclarecer e fornecer algum acesso aos valores de transparência nos campos de cores.

X3D também apresenta uma interface de programação de aplicações (API), diferenciando-se de VRML que apresenta duas API diferentes, uma de script interna e outra externa. Vários problemas encontrados no VRML foram solucionados unificando as duas API, e teve como resultado uma implementação mais confiável e robusta. Conexões via Java já são definidas.

X3D dá suporte a várias codificações de arquivos: VRML, XML e ainda binário comprido. O formato XML permite a aplicação integrar-se com serviços na Web e outros arquivos, e ainda fazer a transferência de dados entre plataformas diferentes. A forma de binário comprido ainda encontra-

se em desenvolvimento, e terá como objetivo fornecer uma maior banda de transmissão de dados. Todas estas codificações dão suporte às características de X3D. Cada uma tem suas vantagens de desvantagens e focam diferentes usos.

Existem muitas mudanças funcionais entre VRML e X3D. As principais estão listadas a seguir.

- A estrutura dos arquivos foi alterada para poder definir as capacidades necessárias no cabeçalho. Para isto, é necessário que seja definido um perfil e qualquer componente extra;
- Os nomes de acesso aos campos que eram eventIn, eventOut, field e exposedField, mudaram agora para inputOnly, outputOnly, initializeOnly e inputOutput, respectivamente;
- Os scripts podem ter campos inputOutput definidos;
- Um nome DEF não pode mais ser definido várias vezes;
- No lugar de existir a necessidade de que os scripts sejam carregados antes de executar o arquivo, em X3D, define-se que o arquivo deve começar primeiro e depois de um certo tempo, com isso os recursos são carregados;
- O modelo de execução do script do grafo de cena é rigorosamente controlado e definido. Enquanto que em VRML um script pode trocar arbitrariamente um grafo de cena em qualquer ponto, em X3D isto só pode ser feito em pontos definidos; e
- O conjunto de definição de tipos abstratos para os nós é rigorosamente definido.

A arquitetura de X3D é modular e por isso provê uma maior flexibilidade e extensibilidade. Grande parte das plataformas existentes não dá suporte a todas as funcionalidades especificadas; muito menos as aplicações desenvolvidas utilizam todos os seus recursos. Por isso, os recursos são agrupados em componentes (*components*). As aplicações dão suporte a estes componentes com o intuito de atingir as necessidades de uma plataforma em particular ou mesmo do mercado. Emo X3D existe ainda o conceito de perfis (*profiles*) que são coleções de componentes com características em comum a um domínio de aplicações, cenários de uso ou plataformas. Um exemplo de um perfil seria os componentes necessários para troca de geometrias entre ferramentas de modelagem. X3D permite diferentes níveis de suporte do padrão com foco em diferentes necessidades, diferentemente de VRML97 que requer um suporte completo de suas funcionalidades. Os componentes de X3D têm a vantagem ainda de poderem ser estendidos de acordo com a vontade do programador, porém seguindo um conjunto de regras rigoroso.

A especificação do X3D tem sofrido mudanças no sentido de se tornar mais flexível para acompanhar as evoluções dos padrões. A especificação de X3D foi dividida em três partes

distintas, são elas: conceitos abstratos e de arquitetura, codificação de arquivo e acesso de linguagem de programação. Esta divisão permite a especificação evoluir mais facilmente. Uma proposta final da especificação de X3D ainda não foi lançada devido às regras de processos ISO.

Não podemos confundir os conceitos e acharmos que um arquivo VRML é o mesmo que um arquivo X3D. iIto não é verdade. Por isso, um navegador puramente X3D não é capaz de ler um arquivo VRML. Existem grandes alterações na sintaxe o que leva à incompatibilidade dos dois padrões. No entanto, grande parte dos navegadores que dá suporte a X3D, também dá suporte a VRML. Isto pode ser verificado na documentação do navegador.

Para converter um arquivo VRML em um arquivo X3D existem diferentes ferramentas disponíveis no mercado. Para simples documentos não há muita complicação, bastando editar o texto manualmente e alterar o cabeçalho para conter a declaração PROFILE. Para documentos complexos é necessário o uso de uma ferramenta auxiliar juntamente com um bom conhecimento da especificação.

4.2.2. Perfis e Componentes

A definição de componentes e perfis é a forma com que X3D define um conjunto de serviços e sua extensibilidade, de acordo com a necessidade do usuário.

Um componente define um conjunto específico de nós com um conjunto de funcionalidades. Ele consiste na definição dos nós e de um conjunto de níveis que englobe um conjunto de aplicações. Um nível simples consiste em uma quantidade pequena de nós e ainda um conjunto de campos limitados. Os níveis maiores consistem de todos os níveis simples com nós mais complexos.

Um perfil é um conjunto de componentes para um nível específico de suporte. Ele não contém outros perfis, no entanto pode utilizar todos os componentes e níveis de outro perfil e adicionar ainda mais. O perfil a ser usado deve ser definido em todos os arquivos X3D. Podemos ainda adicionar mais componentes de acordo com a necessidade do usuário. Ou ainda, definirmos níveis maiores do que os fornecidos pelo perfil ou que não foram definidos ainda.

Novos componentes podem ser criados por cada empresa de acordo com a necessidade dos seus produtos e submete-los para aprovação da diretoria de X3D. Quando submetidos, os componentes sofrem testes e revisões de toda a comunidade X3D, incluindo a diretoria e o Consórcio Web3D. Uma vez aceito e desenvolvido por mais de uma empresa, o prefixo do componente muda para EXT_. Se for classificado pela diretoria recebe então o prefixo X3D_. Se

forem amplamente utilizados e importantes, a diretoria julga se há necessidade de adicioná-lo à especificação ISO oficial.

Uma vez detectada a necessidade de se adicionar novos componentes e/ou perfis à especificação, uma nova versão de X3D é então criada. Uma nova versão implica um maior número de funcionalidades do que a versão anterior. Cada empresa pode criar ainda o seu próprio navegador, suas ferramentas e importadores / exportadores X3D para dar suporte a diferentes versões, componentes e perfis.

4.2.3 Sintaxe

Esta seção visa abordar os principais elementos da sintaxe de X3D. Ela não apresenta toda a especificação. Todos o elementos da especificação de VRML descritos na seção 4.1 também têm suporte em X3D, porém com a escrita diferenciada no formato de XML.

A árvore com os elementos de X3D é bem familiar [9]. Ela tem uma estrutura parecida com HTML, com um cabeçalho e um corpo. O cabeçalho é opcional e somente um é permitido.

Os atributos são mais complexos. O atributo PROFILE (perfil) tem o papel de informar quais as combinações de diferentes produções X3D devem ser ativadas para possibilitar a X3D operar uma variedade de dispositivos, desde *desktop* até um telefone celular.

4.2.3.1. Elemento Head

O tipo de elemento X3D *head* consiste em um componente e declarações metatag. Metatags são os elementos mais utilizados e são um meio de passar uma descrição genérica da informação no documento. Tipicamente conterão dados como nome de autores e datas de publicações. Como abaixo:

```
<!ELEMENT head ( component*, meta* ) >
```

4.3.2.2. Elemento Scene

O elemento *scene* contém o corpo do documento X3D. Neste tipo de elemento, na declaração do tipo do elemento, deixa-se contido aí as declarações dos parâmetros de entidade com o objetivo de apontar o perfil, indicando ao processador X3D pelo valor do atributo do perfil (*profile*) na raiz do documento. Trabalhando com quais produções pertencem a quais componentes e em que perfis é mais vantajoso. Como abaixo:

```
<!ELEMENT Scene  ( %ChildrenNodes; | %WildcardNodes; )*  >
```

4.2.3.3. Elementos Shape

Para expor algo necessitamos usar o elemento *shape*. Como foco geral da sintaxe X3D, apresentaremos apenas um desses elementos, o nó geometria (*geometry*). Enquanto a maioria do performadores de gráficos usam coleções de informações para formar os objetos, X3D inclui um conjunto de formas Euclidianas primitivas – esferas, cubos e cilindros – para possibilitar ao usuário criar e modificar formas básicas. Muito trabalho pode ser feito utilizando apenas formas primitivas. Um exemplo de cena formada apenas por formas primitivas pode ser visto na Figura 8.

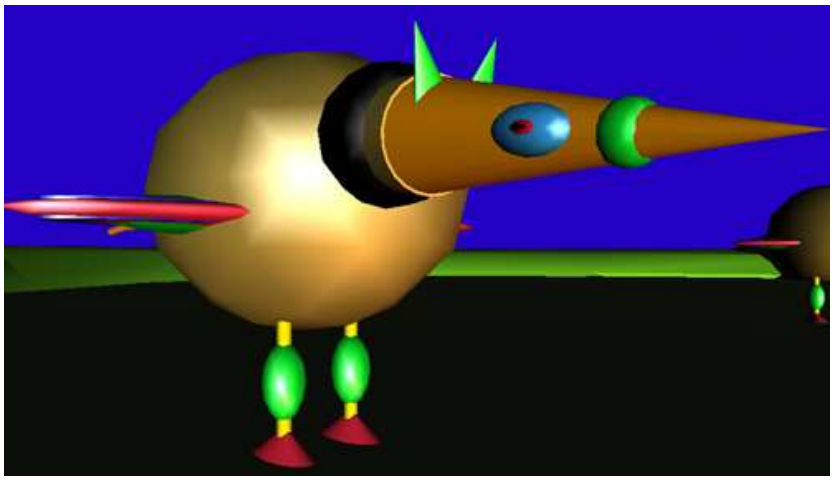


Figura 8 - Exemplo de cena formada apenas por formas primitivas em X3D

4.2.4. Elemento *Appearance*

Cores e texturas são adicionadas à geometria X3D por um tipo de elemento chamado *appearance* (aparência). A declaração do elemento *appearance* é mais complexa que os outros elementos já vistos. Este nó é nomeado utilizando o atributo DEF. Assim como em todos os valores nomeados, isto é útil para reuso e manipulação.

O elemento *appearance* contém um elemento material. Este segundo elemento fornece as informações de cores a ser aplicada à esfera a ser inserida dentro do elemento *shape* (forma). Pode-se ainda utilizar um elemento *ImageTexture* (textura de imagem) ou *MovieTexture* (textura de filme) ao invés do elemento material para obter resultados diferentes. A Figura 9 exibe um exemplo de uma forma com textura de imagem.

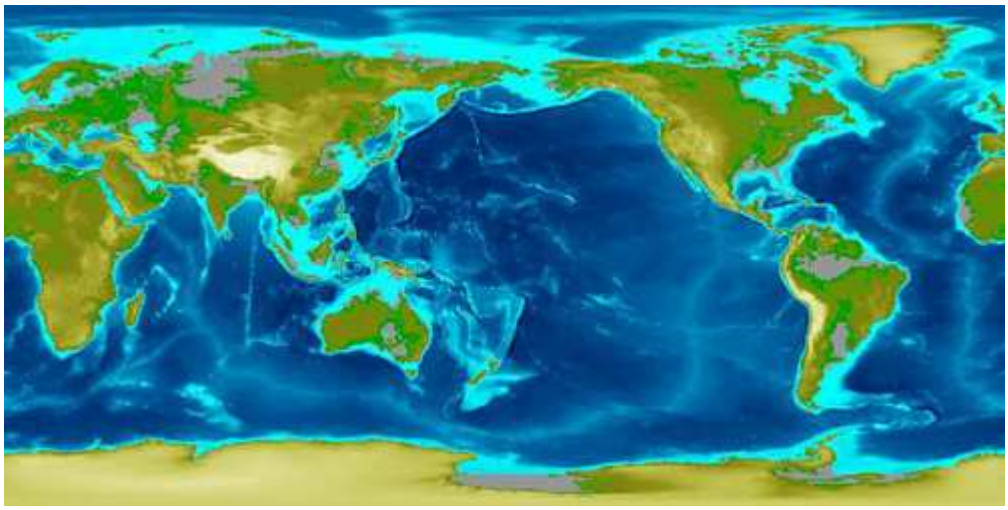


Figura 9 - Forma com textura de imagem

O modelo de gerenciamento de mídias e objetos virtuais proposto por este trabalho utiliza a textura de filme para executar no bowser a mídia armazenada num banco de dados.

5. Estudo de Caso/Protótipo

Este capítulo detalha os requisitos, a modelagem e o desenvolvimento do módulo de gerenciamento de mídias e objetos virtuais, o MidiaWEB. Discutiremos ainda as decisões de projeto e as limitações do protótipo.

A seção 5.1 descreve os requisitos necessários para atingir os objetivos do MidiaWEB. Na seção 5.2 descrevemos as decisões do projeto. A seção 5.3 apresenta o diagrama de casos de uso e a arquitetura escolhida para a implementação do sistema. Finalmente, a seção 5.4 discute as dificuldades encontradas no desenvolvimento e os resultados obtidos.

5.1. Requisitos

O objetivo geral do MidiaWEB é desenvolver uma infra-estrutura para gerenciamento de objetos virtuais e mídias. Estes objetos e mídias devem ser carregados a partir de cenas já desenvolvidas anteriormente e armazenados em um SGBD XML. Ele provê serviços de armazenamento e recuperação desses objetos.

O MídiaWEB apresenta as seguintes funcionalidades:

- Inserir mídia;
- Inserir de objeto virtual;
- Listar mídias;
- Listar objetos virtuais;
- Alterar objeto virtual;
- Visualizar mídia; e
- Visualizar objeto virtual.

Para inserir uma mídia ou um objeto virtual, o usuário indica o caminho para o arquivo. Em seguida o programa recupera este arquivo e o armazena no SGBD Oracle 10g.

Na listagem de mídias e objetos virtuais o sistema recupera o nome dos arquivos inseridos no banco e os lista na tela. É então disponibilizada a opção para visualização do arquivo. No caso de objetos virtuais além da visualização é disponibilizada também a opção para alteração.

Para alterar um objeto virtual o usuário deverá escolher um dos objetos disponíveis para inserir na cena. O sistema então faz a alteração diretamente no banco de dados. Para visualizar a alteração o usuário deve entrar novamente na função de listagem.

Finalmente, a visualização das mídias e dos objetos virtuais procede quando o usuário dá um clique no *mouse* apontando o link para o arquivo. É gerada então a imagem de exibição. No caso das mídias, quando a mídia é recuperada do banco, ela é inserida em um arquivo do tipo X3D para a sua execução na tela.

5.2. Decisões de Projeto

Quanto ao SGBD a ser utilizado, o próprio trabalho dá ênfase ao Oracle 10g. Por isso, este foi o escolhido para fazer o armazenamento dos dados. O Oracle oferece muitos recursos para manipulação de documentos XML, como foi visto na Seção 5 do Capítulo 3.

O protótipo foi implementado na linguagem Java, utilizando a tecnologia JSP [13]. Foi utilizado ainda o servidor Tomcat [3]. Para exibir o conteúdo de mídia e de objeto virtual foi utilizado o *plug-in* Flux Player [10].

O armazenamento das mídias e dos objetos virtuais deu-se de formas distintas. O objeto virtual foi armazenado em um campo do tipo XMLType, já o arquivo de mídia foi armazenado em um campo do tipo BLOB (Binary Large Object).

5.3. Modelagem

A Figura 10 mostra o Diagrama de Casos de Uso do MidiaWEB, considerando os serviços descritos na Seção 1 deste capítulo

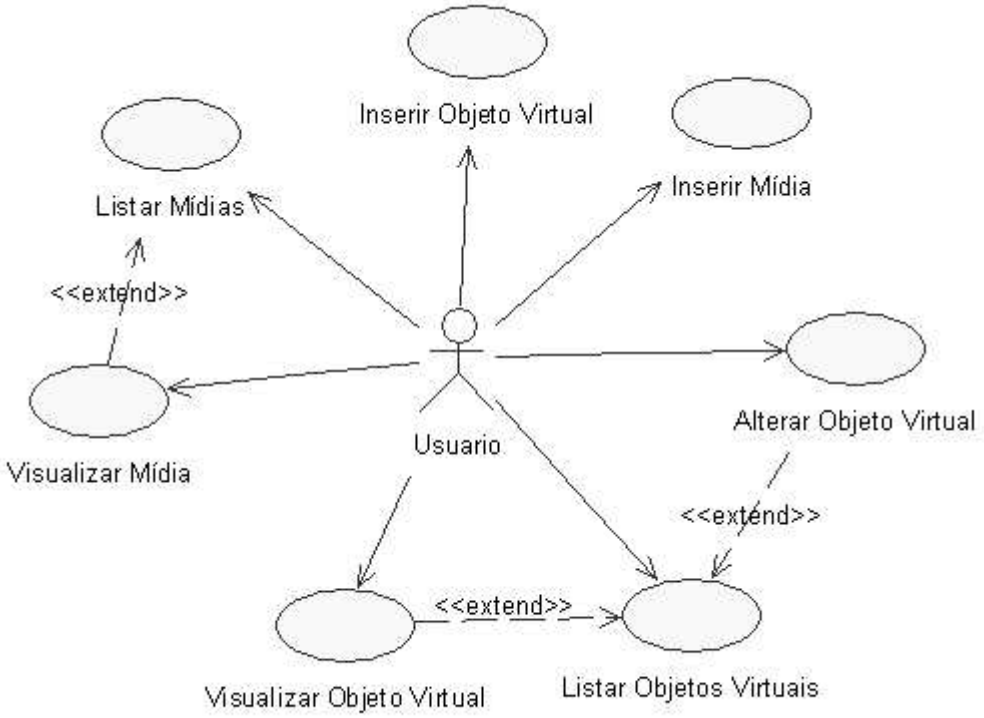


Figura 10 – Casos de Uso do MidiaWeb

A Figura 11 mostra o Diagrama de Classes do MidiaWEB. A classe *RepositorioOracle* é responsável em fazer o acesso ao SGBD. Ela implementa as funcionalidades da classe *IRepositorio*. A classe Fachada é quem recebe as solicitações do código em JSP e as processa.

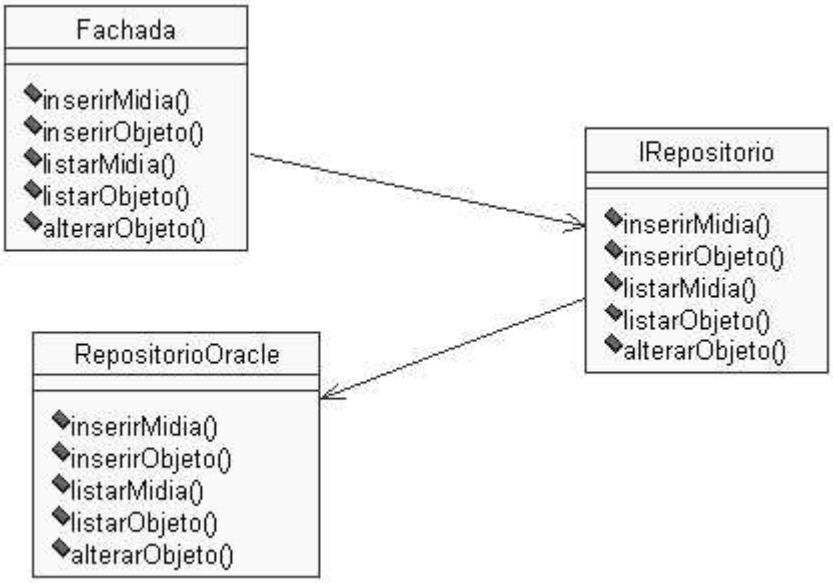


Figura 11 – Diagrama de Classes MidiaWeb

5.4. Resultados

Os objetivos do MidiaWeb foram alcançados. A funcionalidade especificada no seu projeto foi implementada integralmente. Algumas dificuldades foram encontradas no uso do Tomcat juntamente com o plug-in Flux Player. A Figura 12 apresenta a tela principal do sistema. Todas as telas têm acesso às funções de inserir mídia, inserir objeto virtual, listar mídia e listar objeto virtual.

Figura 12 – Tela Principal

A Figura 13 apresenta a tela de inserção de mídia e de objeto virtual. O usuário deve selecionar com o *mouse* o botão “procurar” e especificar o caminho para o arquivo que será inserido no banco. Em seguida, deve selecionar o botão “cadastrar” para que seja salvo no banco.

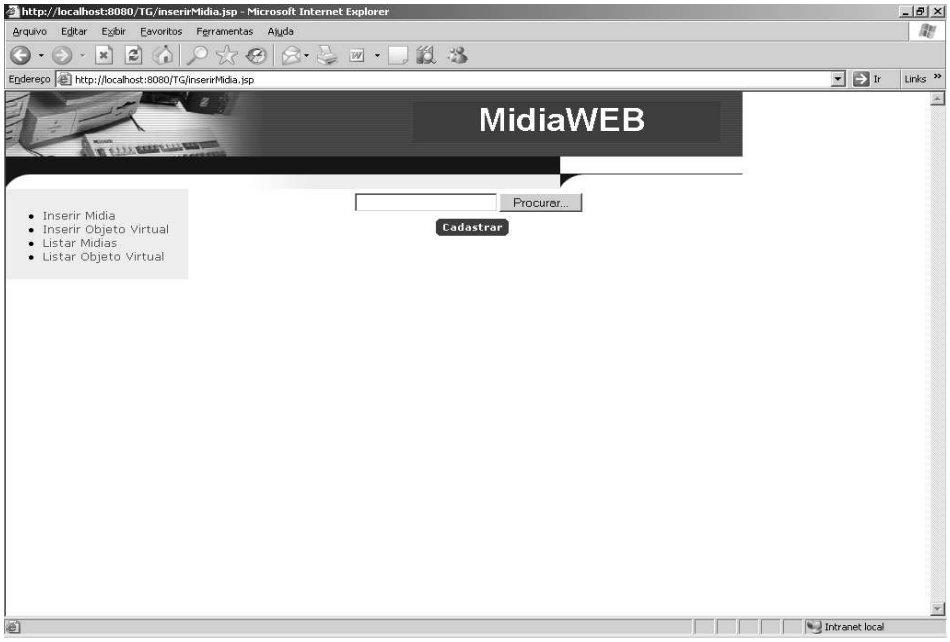


Figura 13 – Inserção de Objeto Virtual e de Mídias

Para visualizar a lista de mídias cadastradas no banco o usuário deve selecionar com o *mouse* em “Listar Mídias”. Como resultado, é apresentada a tela na Figura 14. A partir daí, o usuário escolhe qual das mídias quer visualizar e em seguida seleciona o botão “visualizar”.

Figura 14 – Listar Mídias

Quando a opção de visualizar é acionada, o sistema abre então a tela apresentada na Figura 15. Ela apresenta um *link* para o arquivo a ser exibido.

Figura 15 – Link para Visualização do Arquivo

A visualização dos objetos virtuais acontece de maneira semelhante. Primeiro o usuário escolhe a opção de “Listar objetos virtuais” e a tela da Figura 16 é apresentada. O usuário escolhe então qual dos objetos deseja visualizar e pressiona o botão “visualizar”. A tela da Figura 17 é então apresentada com o *link* para o arquivo. Selecionando o *link*, o sistema exibe então o objeto virtual que pode ver visualizado na Figura 18.

Figura 16 – Lista de Objetos Virtuais

Figura 17 – Link para Visualização do Objeto Virtual

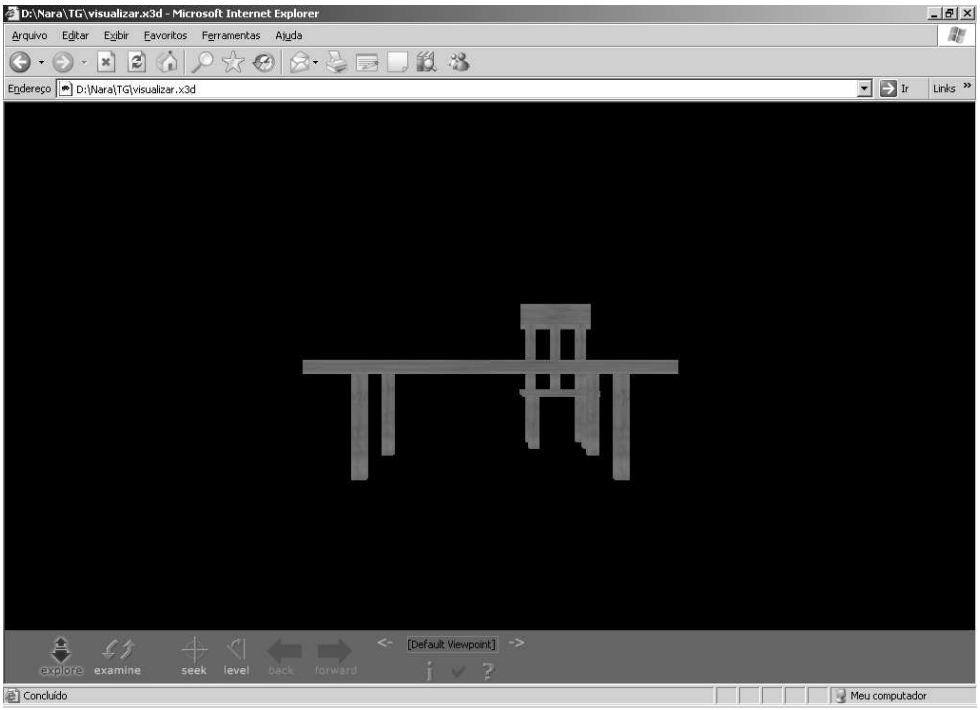


Figura 18 – Exibição do Objeto Virtual

A função de alteração do ambiente virtual limitou-se apenas a acrescentar um objeto do tipo cadeira à cena. A partir da tela da Figura 16, o usuário pode escolher a opção de “adicionar cadeira”. Esta função altera o arquivo XML escolhido e o salva no banco. Para visualizar as alterações o usuário deve listar novamente todos os objetos virtuais e em seguida selecioná-lo para visualização.

6. Conclusão

A disseminação da internet tem incentivado cada vez mais a construção de ferramentas de manipulação de mídias para exibição na Web. Entretanto, os ambientes atuais apresentam algumas limitações. Uma delas é a inexistência de mecanismos de reutilização de objetos de um BD.

Este trabalho teve como objetivo principal a construção de um módulo de gerenciamento de objetos de realidade virtual e de mídias que dê apoio a sua exibição na Web. Neste contexto, foi construído o MidiaWEB que agregou diversos serviços de armazenamento e gerenciamento de mídias e objetos virtuais. A criação do caso de teste do Capítulo 4 permitiu demonstrar que o MidiaWEB pode ser um elemento importante para a exibição de mídias na Web.

Durante a construção deste módulo ocorreram alguns problemas que impediram que mais recursos fossem adicionados ao sistema. A primeira delas diz respeito à inexperiência da aluna em lidar com a manipulação de documentos XML, especialmente em conjunto com um SGBD. A segunda se refere àineficiência do plug-in escolhido juntamente com o Tomcat para reconhecer a execução de um ambiente virtual.

Como futuros trabalhos, uma sugestão é realizar testes mais rigorosos com mídias diferenciadas e mais pesadas, como filmes longos, por exemplo. Outra sugestão é portar o sistema para SGBD diferentes do Oracle, em especial para SGBD gratuitos, o que possibilitará o uso do MidiaWeb por um número muito maior de usuários.

7. Referências Bibliográficas

[1] A Linguagem XML e Especificações Associadas
Artur Afonso de Sousa
FCA - Editora de Informática

[2] An Introduction to VRML
Tuomas Lukka
URL: http://www.linuxjournal.com/article/3085

[3] Apache Tomcat
URL: http://tomcat.apache.org/

[4] Berkeley DB XML
URL: http://www.sleepycat.com/products/bdbxml.html

[5] CVS - Concurrent Versions System
URL: http://www.cvshome.org

[6] Document Type Definition
URL: http://www.w3.org/TR/REC-html40/sgml/dtd.html

[7] Encyclopedia – PC Magazine
URL: http://www.pcmag.com/encyclopedia/

[8] Extensible Markup Language (XML)
URL: http://www.w3.org/XML/

[9] Extensible 3D: XML Meets VRML
URL: http://www.xml.com/pub/a/2003/08/06/x3d.html?page=3

[10] Flux Player
URL: http://www.mediamachines.com/playerproductpage.html

[11] Introdução a XML
URL: http://www.criarweb.com/xml/

[12] Introduction to Native XML Databases
URL: http://www.xml.com/lpt/a/2001/10/31/nativexmldb.html

[13] JSP - JavaServer Pages
URL: http://java.sun.com/products/jsp/

[14] MS-DOS
URL: http://www.microsoft.com/mspress/brasil/subjects/subjectws_p1.htm

[15] Oracle Database
URL: http://www.oracle.com/database

[16] Oracle Database 10g Relase 2 XML DB Technical Overview
Geoff Lee
Oracle Corporation, Maio 2005

[17] SAX
URL: http://www.saxproject.org/

[18] SQL 2003
URL:
[19] SGBD Tamino
URL: Extensible Stylesheet Language Family Transformations

[20] Tendências em XML
Helder da Rocha
URL: http://www.argonavis.com.br/palestras/od2002/OD_XML.pdf

[21] Tutorial XML
Miguel Benedito Furtado Júnior
URL: http://www.gta.ufrj.br/grad/00_1/miguel/index.html

[22] Uma Visão Geral sobre XML
Ronaldo dos Santos Mello
Departamento de Informática e Estatística (INE) – Centro Tecnológico (CTC)
Universidade Federal de Santa Catarina (UFSC)

[23] Unix
URL: http://www.unix.org

[24] VRML
URL: http://www.web3d.org/x3d/vrml/

[25] Web3D Consortium
URL: http://www.web3d.org/

[26] What is the Document Object Model?
URL: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/introduction.html

[27] W3C Document Object Model
URL: http://www.w3.org/DOM/

[28] XML and Databases
URL: http://www.rpbourret.com/xml/XMLAndDatabases.htm

[29] XML Schema
URL: www.w3.org/XML/Schema

[30] XML: DB Initiative: Projects
URL: http://xmldb-org.sourceforge.net/projects.html

[31] XML: saiba o que é
URL: http://www.infowester.com/lingxml.php

[32] XSLT- Extensible Stylesheet Language Family Transformations
URL: http://www.w3.org/TR/xslt

[33] XML Tutorial
URL: http://www.w3schools.com/xml/]

Assinaturas

_____

Nara de Arruda Falcão
Aluna

_____

Fernando da Fonseca de Souza
Orientador