

Servidor de Documentos XML Usando Java

Werley Ribeiro Martins

Orientador

Prof. Dr. Dilvan de Abreu Moreira

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação, da Universidade de São Paulo - USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área de Ciências de Computação e Matemática Computacional.

São Carlos
Agosto de 2001

Agradecimentos

Primeiramente, a Deus, por sempre me dar força diante das dificuldades e por ter conseguido dar mais um passo na caminhada da minha vida.

À minha família, em especial aos meus pais. Mesmo estando longe, vocês nunca deixaram de me incentivar e de acreditar em mim. Obrigado a todos vocês!

Ao professor Dilvan, pela oportunidade e pelas contribuições dadas durante o desenvolvimento deste trabalho.

Aos amigos das repúblicas por onde passei. Além do convívio diário, foi a família que me acolheu durante todo esse tempo de mestrado.

A todos aqueles com quem convivi durante esse tempo, pelas muitas alegrias que passei, as quais estarão sempre guardadas na lembrança.

A todos que, de alguma forma, colaboraram e fizeram parte para que este trabalho pudesse ter sido realizado. Em especial, àqueles que me deram força até os momentos finais da conclusão do trabalho, pois estes sempre confiaram em mim e nunca me abandonaram.

A Capes, pelo apoio financeiro.

Sumário

CAPÍTULO 1 - INTRODUÇÃO	1
1.1 - CONSIDERAÇÕES INICIAIS.....	1
1.2 - MOTIVAÇÃO.....	3
1.3 - OBJETIVO	3
1.4 - ORGANIZAÇÃO DO TRABALHO	4
CAPÍTULO 2 - LINGUAGENS DE MARCAÇÃO.....	6
2.1 - CONSIDERAÇÕES INICIAIS.....	6
2.2 - SGML	7
2.3 - HTML	8
2.4 - XML	11
2.4.1 - DTD	14
2.4.2 - Linguagens de Estilos.....	16
2.4.3 - XQL.....	20
2.5 - CONSIDERAÇÕES FINAIS.....	23
CAPÍTULO 3 - REPOSITÓRIOS DE DADOS.....	25
3.1 - CONSIDERAÇÕES INICIAIS.....	25
3.2 - SISTEMA DE BANCO DE DADOS RELACIONAL.....	26
3.3 - JAVASPACEs	27
3.3.1 - Definição	28
3.3.2 - Operações.....	30

3.3.3 - <i>Vantagens de JavaSpaces</i>	31
3.4 - BANCO DE DADOS RELACIONAIS E <i>JAVASPACEs</i>	32
3.5 - CONSIDERAÇÕES FINAIS.....	33
CAPÍTULO 4 - ACESSO E MANIPULAÇÃO DE DADOS XML	34
4.1 - CONSIDERAÇÕES INICIAIS.....	34
4.2 - A LINGUAGEM DE PROGRAMAÇÃO JAVA	35
4.3 - SAX	36
4.4 - DOM.....	38
4.5 - BIBLIOTECAS.....	40
4.5.1 - <i>Xerces</i>	41
4.5.2 - <i>Xalan</i>	42
4.5.3 - <i>GMD-IPSI XQL</i>	43
4.6 - CONSIDERAÇÕES FINAIS.....	44
CAPÍTULO 5 - O SERVIDOR XML	45
5.1 - CONSIDERAÇÕES INICIAIS.....	45
5.2 - DEFINIÇÃO E TECNOLOGIAS UTILIZADAS.....	45
5.3 - A INTERFACE DO SERVIDOR XML	48
5.4 - FUNCIONAMENTO DOS MÉTODOS DA INTERFACE.....	50
5.5 - UM CLIENTE GRÁFICO	56
5.6 - TESTES REALIZADOS.....	62
5.7 - CONSIDERAÇÕES FINAIS.....	63
CAPÍTULO 6 - CONCLUSÕES	64
6.1 - CONSIDERAÇÕES INICIAIS.....	64
6.2 - RESULTADOS E CONTRIBUIÇÕES	64
6.3 - SUGESTÕES PARA TRABALHOS FUTUROS.....	65
REFERÊNCIAS BIBLIOGRÁFICAS.....	67

Lista de Tabela e de Figuras

Tabela 2.1 – Comparação entre SQL e XQL (Robie, 1999b).	21
Figura 2.1 – Um documento HTML.....	9
Figura 2.2 – Representação gráfica do documento HTML da Figura 2.1.	10
Figura 2.3 – Um documento XML.....	12
Figura 2.4 – Sistema baseado em XML (McGrath, 1999).	13
Figura 2.5 – Um documento DTD referenciado por um documento XML.....	15
Figura 2.6 – Verificação da estrutura de um documento XML com um parser XML (McGrath, 1999).	15
Figura 2.7 – Transformação de um documento XML em outros formatos usando a XSL (McGrath, 1999).	17
Figura 2.8 – Um documento XSL.....	18
Figura 2.9 – Uma consulta XQL.....	22
Figura 3.1 – Processos interagindo com espaços diferentes (JAVASPACES, 2000b). ..	29
Figura 4.1 – Representação do documento XML da Figura 2.2 através de SAX.	37
Figura 4.2 – Representação do documento XML da Figura 2.2 através de DOM.	39
Figura 5.1 – Arquitetura do Servidor XML.....	48

Figura 5.2 – Interface do Servidor XML.	49
Figura 5.3 – Ilustração da representação DOM de um documento XML armazenado no espaço.	52
Figura 5.4 – Ilustração da representação DOM do novo documento XML.	52
Figura 5.5 – Representação DOM do documento após a operação addIn.	53
Figura 5.6 – Representação DOM do documento após a operação delete.	54
Figura 5.7 – Representação DOM do documento após a operação de consulta.....	55
Figura 5.8 – Conexão com o servidor XML.....	57
Figura 5.9 – Funções contidas na aplicação cliente.	57
Figura 5.10 – Processo para adicionar um documento.	58
Figura 5.11 – Processo para apagar dados de um documento.	59
Figura 5.12 – Processo para consultar dados de um documento.	60
Figura 5.13 – Processo para transformação de um documento.	61
Figura 5.14 – Processo para manipulação com names.	62

Resumo

A linguagem XML (*Extensible Markup Language*) vem sendo muito utilizada em diversas aplicações e já é reconhecida como uma linguagem padrão para representação de documentos tanto no meio acadêmico quanto no comercial. Porém, a disponibilidade de uma grande quantidade de dados apresenta vários problemas práticos que o padrão XML, por si só, não pode resolver. O objetivo deste trabalho é desenvolver um servidor XML capaz de realizar diversas operações (como adicionar, consultar, apagar e transformar) sobre um repositório de documentos XML e XSL (*Extensible Stylesheet Language*). Esses documentos são representados na forma de objetos Java que obedecem ao modelo de interfaces DOM (*Document Object Model*), um modelo padrão usado em várias aplicações da Internet. Para o armazenamento é utilizado o serviço *JavaSpaces*, que pode ser usado como um repositório persistente para esses objetos DOM-Java. O servidor XML utiliza a linguagem XQL (*XML Query Language*) na realização de consultas. Além disso, ele permite, através da linguagem XSL, a conversão de documentos XML para outras linguagens.

Abstract

The Extensible Markup Language (XML) has been used in several applications and it is being already recognized as a standard language for document representation in the academic and business worlds. However, great amounts of data present several practical problems that the XML standard itself does not address. The goal of this project was to develop a XML server capable of doing many kinds of operations (such as adding, querying, deleting and transforming) over a repository of XML and XSL (Extensible Stylesheet Language) documents. These documents are represented using Java objects conforming to the DOM (Document Object Model) interface model, a standard model used in many Internet applications. For object storage, the JavaSpaces service is used as a persistent repository for the DOM-Java objects. The XML server uses the XQL (XML Query Language) as query language. It also allows the conversion of XML documents into other languages using the XSL language.

Capítulo 1 - Introdução

1.1 - Considerações Iniciais

Sistemas hipermídia são caracterizados por aplicações nas quais usuários navegam interativamente por documentos. O ambiente hipermídia WWW (*World Wide Web*) se caracteriza por disponibilizar milhões de hiperdocumentos interligados possuindo diversos tipos de dados. Além disso, no WWW é possível encontrar um grande número de aplicações cliente/servidor disponíveis, onde usuários podem interagir continuamente.

Nos últimos anos, vem ocorrendo um grande crescimento de aplicações no ambiente WWW, devido ao baixo custo e à facilidade de distribuição de documentos eletrônicos a um grupo de usuários cada vez maior. Ao mesmo tempo, a tecnologia da Internet vem aumentando a demanda por aplicações comerciais que sejam mais intuitivas, amigáveis e dinâmicas.

Com o aumento do número de usuários de servidores no WWW, ao invés de trabalhar com servidores individuais, os usuários podem usar a Internet como um grande disco virtual contendo qualquer tipo de informação desejada e tudo disponível ao simples clicar de um *mouse*.

Com esse crescimento, os documentos eletrônicos estão tornando-se cada vez mais complexos, gerando, com isso, uma grande limitação na sua manipulação, falta de extensibilidade e intercâmbio com outras aplicações (Bosak, 1997).

O padrão utilizado para formatar os documentos apresentados no WWW é definido pela linguagem HTML (*HyperText Markup Language*), que foi criada a partir da metalinguagem SGML (*Standard Generalized Markup Language*) (ISO, 1986). Apesar de ser uma linguagem simples e de fácil entendimento, a HTML não é totalmente adequada para suportar a variedade de aplicações executadas sobre o WWW. Suas principais desvantagens são a falta de suporte à estrutura do documento, o que limita a consulta apenas a trechos de texto, e a não extensibilidade, impedindo os desenvolvedores de criarem seus próprios tipos de elementos (*tags*) para indicar o significado do conteúdo do documento (Bosak, 1997, Bray et al., 2000). Já a linguagem SGML, por sua vez, é uma linguagem muito poderosa e que não possui tais desvantagens, porém é muito complexa para ser utilizada diretamente na definição de novas classes de documentos WWW (Cleveland, 1998).

Sendo assim, foi proposta uma nova linguagem, para a definição de documentos estruturados, que tivesse a extensibilidade que falta à HTML e que fosse mais simples que a SGML, sendo chamada de XML (*Extensible Markup Language*) (Bray et al., 2000). O objetivo da XML é fornecer muito dos benefícios encontrados em SGML e não disponíveis em HTML (Mace et al., 1998; McGrath, 1999).

A grande vantagem da utilização de documentos XML em relação aos documentos HTML é a separação entre o conteúdo e a apresentação. Desse modo, é possível alterar o conteúdo do documento sem se preocupar com sua forma de apresentação. Além disso, com o uso da XML, os projetistas podem criar seus próprios tipos de elementos de acordo com a aplicação que está sendo projetada e, com isso, indicar o significado dos dados.

Como a XML não especifica aspectos de apresentação sobre seus elementos, eles podem utilizar uma folha de estilos usada para formatar sua apresentação quando necessário. Dessa forma, foi criada a linguagem XSL (*Extensible Stylesheet Language*) para ser usada com o propósito de expressar como o conteúdo dos documentos XML deve ser apresentado. A XSL tem ainda a propriedade de ser independente de qualquer tipo de formato de apresentação resultante (Adler et al., 1997). Com isso, um documento XML pode ser transformado, usando-se a linguagem XSL, em outros formatos de documento como RTF, TeX, *PostScript* HTML.

1.2 - Motivação

A linguagem XML traz muitas promessas para o futuro, já sendo reconhecida hoje como uma linguagem para representação da estrutura de documentos tanto no meio acadêmico quanto no meio comercial. No entanto, a disponibilidade de uma grande quantidade de dados XML apresenta várias questões técnicas que o padrão XML não discute, tais como (Deutsch et al., 1998):

- Como os dados XML podem ser extraídos de grandes documentos XML?
- Como os dados XML podem ser trocados entre vários usuários através de especificações de conceitos diferentes, porém relacionadas?
- Como os dados XML de diversas fontes XML podem ser integrados?

Para isso, foi criada a linguagem XQL (*XML Query Language*), que é uma linguagem de consulta projetada especificamente para documentos XML. A XQL é usada para endereçar e filtrar os elementos e texto de documentos XML, permitindo consultar qualquer tipo de dado em XML (Robie et al., 1998).

Entretanto, apesar de existir uma linguagem que permite consultar dados em XML, deseja-se, a partir daí, realizar algum tipo de manipulação com esses dados. Dessa forma, a motivação principal para o desenvolvimento deste trabalho é poder executar operações sobre dados de documentos XML retornados por uma linguagem de consulta, no caso desse projeto através da linguagem XQL. Assim, seria possível realizar diversos tipos de manipulações sobre dados de documentos XML, tornando mais fácil o trabalho do usuário. Um exemplo desse tipo de operação seria retirar somente uma determinada parte de um documento XML, editá-la e depois adicioná-la novamente no mesmo lugar ou, então, em qualquer outra parte do documento original.

1.3 - Objetivo

O objetivo deste trabalho é desenvolver um servidor XML capaz de realizar diversas operações (como adicionar, consulta, apagar e transformar) sobre um repositório de documentos XML e XSL.

Esses documentos são representados na forma de objetos que obedecem ao modelo de interfaces DOM (*Document Object Model*), um modelo padrão usado em várias aplicações da Internet. Para o armazenamento dos documentos XML e XSL na forma de objetos Java compatíveis com DOM é utilizado o serviço *JavaSpaces*, que pode ser usado como um repositório compartilhado onde os objetos Java podem ser armazenados de forma persistente (Freeman et al., 1999).

O servidor utiliza a linguagem XQL na realização de consultas. Além disso, ele permite, através da linguagem XSL, a conversão de documentos XML para outras linguagens definidas de acordo com especificações contidas em documentos XSL.

1.4 - Organização do Trabalho

Essa seção apresenta o conteúdo dos demais capítulos deste trabalho desenvolvido de acordo com a descrição que se segue abaixo.

O Capítulo 2 trata das linguagens de marcação SGML, HTML e XML, descrevendo suas características principais, bem como suas vantagens e desvantagens. Ele inclui também os conceitos de DTD (*Document Type Definition*) e de linguagens de estilo, como é o caso da linguagem XSL. No final, é abordada a linguagem de consulta XQL, sendo feita uma comparação com a SQL (*Structured Query Language*), que é uma linguagem de consulta padrão para bancos de dados relacionais.

No Capítulo 3, são descritas duas formas diferentes de se realizar o armazenamento e o acesso a informações. Tais formas são definidas através de banco de dados relacionais e da tecnologia *JavaSpaces*, que pode ser usada para o armazenamento de documentos XML de forma persistente. Também é feita uma comparação de algumas das principais diferenças entre esses repositórios.

Já no Capítulo 4, são abordadas as formas de acesso aos dados XML. Inicialmente, são apresentadas as principais características da linguagem Java, que é a linguagem de programação escolhida para o desenvolvimento deste trabalho. Também são descritas algumas das principais interfaces DOM, bem como as vantagens e desvantagens de sua utilização em comparação com SAX (*Simple API for XML*), que é uma outra forma de acesso aos dados contidos em documentos XML. Além disso, são descritas as bibliotecas Xalan, Xerces e GMD-IPSI XQL. Essas bibliotecas, escritas em

Java, são utilizadas para realizar as manipulações com os dados XML e utilizam as interfaces DOM em suas implementações.

No Capítulo 5, é discutido o desenvolvimento do servidor XML, descrevendo toda sua funcionalidade, desde a sua definição e as tecnologias utilizadas, passando pela descrição dos métodos desenvolvidos em sua interface e como esses métodos trabalham, apresentando ainda uma interface gráfica desenvolvida para ilustrar o funcionamento desses métodos, além dos testes realizados, para garantir o bom funcionamento do servidor.

Por fim, o Capítulo 6 refere-se à conclusão a respeito deste trabalho, além das contribuições e das sugestões para trabalhos futuros.

Capítulo 2 - Linguagens de Marcação

2.1 - Considerações Iniciais

Ao observar a estrutura de um livro, verifica-se que este possui um título, um índice e vários capítulos, estes por sua vez possuem um título, uma ou mais seções com um ou mais parágrafos. Já a estrutura de uma carta não possui nenhum dos elementos citados no livro, porém, ela possui data, saudação, conteúdo e assinatura. Essa estrutura varia significativamente entre diversos tipos de documentos, tais como formulários, documentos bancários, menus de restaurantes entre outros.

A marcação de documentos serve para identificar a estrutura e o formato no qual o documento deve ser apresentado, de modo a obter uma boa aparência na apresentação final. Uma marcação pode indicar que uma frase tenha tamanho de letra diferente, destacar uma palavra colocando-a em negrito, ser uma quebra de página entre outros, sendo que cada documento tem o seu conjunto de marcações e apresentador próprio.

A partir dessa idéia, alguns pesquisadores procuraram um padrão de representação de documentos que pudesse ser utilizado em todas as estruturas que se fizessem necessárias para maior facilidade na criação dos documentos. Com esse intuito, surgiram as linguagens de marcação.

Neste capítulo, são abordadas as principais características das linguagens de marcação SGML (*Standard Generalized Markup Language*), HTML (*HyperText Markup Language*) e XML (*Extensible Markup Language*). Também são detalhados os conceitos de DTD (*Document Type Definition*) e de linguagens de estilo. Por fim, é

apresentada a linguagem XQL (*XML Query Language*), uma linguagem de consulta projetada exclusivamente para documentos XML.

2.2 - SGML

A linguagem SGML (*Standard Generalized Markup Language*) foi proposta para permitir a definição de documentos de acordo com sua estrutura e conteúdo, independente de sua apresentação (ISO, 1986). É uma linguagem genérica para a descrição da estrutura lógica de documentos, permitindo a definição de linguagens específicas, sendo que essas últimas definem a estrutura hierárquica de um tipo de documento. O padrão SGML deu origem a outros padrões que surgiram com a demanda do WWW (*World Wide Web*) por novos recursos.

Quando usados em aplicações apropriadas, os benefícios da SGML, que garantiram a sua aceitação e que até hoje permanecem, são bastante significantes. Eles incluem (Cleveland, 1998):

- Estrutura hierárquica - permite a representação da estrutura hierárquica dos elementos em um documento, facilitando o seu processamento.
- Flexibilidade - não especifica quais tipos de elementos devem ser criados ou como eles se relacionam, ficando isso a critério do usuário. Assim, é possível criar vários tipos de documentos diferentes.
- Especificação formal - todos os elementos contidos em um documento SGML são declarados antes de ele começar. O programa que utiliza o documento SGML processa esse documento juntamente com sua declaração, permitindo validá-lo.
- Representação legível - um documento SGML tem conteúdo textual, podendo ser manipulado em qualquer editor ou lido e compreendido com facilidade por um usuário.
- Reutilização da informação - um documento SGML contém componentes identificados exclusivamente (parágrafos, seções, etc.) que podem ser reutilizados facilmente em outros documentos.

- Vários níveis de segurança - como os componentes de um documento SGML são identificados exclusivamente, pode-se ter diferentes níveis de segurança para cada componente.

A linguagem SGML é um padrão muito poderoso e geral e tem sido utilizada em ramos como o da publicação técnica, indústrias farmacêuticas, companhias aeroespaciais, automotivas e de telecomunicações. Mas apesar dos benefícios que podem ser ganhos usando SGML, sua base de usuários foi limitada a grandes empresas, pois se trata de uma linguagem muito complexa, fazendo com que os custos com implementação não sejam triviais. Desse modo, as idéias que a SGML incorpora são simplesmente muito significantes e úteis para ficarem restritas a tal nicho. Em (Cleveland, 1998) pode-se obter outras informações sobre as principais áreas onde ocorrem custos, como treinamento e investimento em novos *software* e *hardware*, devido à complexidade da SGML.

2.3 - HTML

A linguagem HTML (*HyperText Markup Language*) surgiu devido à necessidade de uma linguagem simples e reduzida para especificar a estrutura e a apresentação dos documentos disponíveis no WWW, sendo um fato que contribuiu para a expansão da Internet (Raggett et al., 1999). Além disso, ela foi criada a partir da metalinguagem SGML e um de seus objetivos principais é formatar esses documentos na forma como eles devem ser apresentados.

A HTML tornou-se amplamente utilizada como um formato para especificação de documentos hipermídia por sua simplicidade e facilidade com que os documentos podem ser escritos. A linguagem especifica um modelo para documentos textuais contendo interligações e objetos em outras mídias. Além disso, devido à simplicidade do seu código e à limitação no número de elementos que a constitui, a HTML permite que seu uso seja simplificado e que qualquer usuário com conhecimentos básicos em computação possa criar um hiperdocumento definido nessa linguagem.

Ainda em HTML é definido um conjunto de tipos de elementos (*tags*) utilizados para especificar a informação e também para indicar o formato com o qual essa

informação deve ser apresentada. A **Figura 2.1** mostra um exemplo de um documento HTML.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<TITLE>Servidor de Documentos XML Usando JAVA</TITLE>
</HEAD>
<BODY VLINK="#551A8B" TEXT="#000000" LINK="#0000EE" BGCOLOR="#FFFFFF"
ALINK="#FF0000">
<H1 ALIGN="CENTER">Servidor de Documentos XML Usando JAVA</H1>
<BR>
<BR>
<H2 ALIGN="LEFT">Capítulo 1 - Introdução</H2>
<H3 ALIGN="LEFT">1.1 - Considerações Iniciais</H3>
<DD>
<P ALIGN="JUSTIFY">...</P>
</DD>
<DD>
<P ALIGN="JUSTIFY">Com esse crescimento, os documentos eletrônicos estão
tomando-se cada vez mais complexos, gerando, com isso, uma grande limitação
na sua manipulação e falta de extensibilidade e intercâmbio com outras
aplicações (Bosak, 1997).</P>
</DD>
<DD>
<P ALIGN="JUSTIFY">...</P>
</DD>
<BR>
<H3 ALIGN="LEFT">1.2 - Motivações</H3>
<DD>
<P ALIGN="JUSTIFY">...</P>
</DD>
<BR>
</BODY>
</HTML>
```

Figura 2.1 – Um documento HTML.

A apresentação gráfica do documento HTML da **Figura 2.1** em um *browser* pode ser vista na **Figura 2.2**. A explicação de cada elemento que compõe esse documento não pertence ao escopo deste trabalho, de modo que essas informações podem ser obtidas em (Raggett et al., 1999).

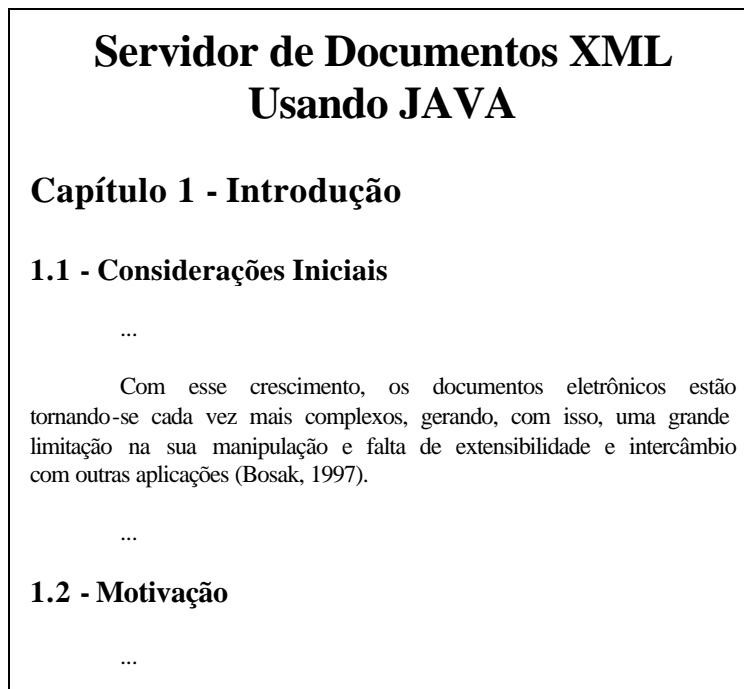


Figura 2.2 – Representação gráfica do documento HTML da Figura 2.1.

Apesar da linguagem HTML ser muito simples e utilizada em diversas aplicações, existem algumas desvantagens relacionadas à sua flexibilidade como (Bosak, 1997; Johnson, 1999; Mace et al., 1998):

- ser uma linguagem que representa a informação em termos de seu *layout* e não através do seu significado;
- fornecer uma única maneira de visualizar os mesmos dados e se houver a necessidade de mudanças na forma de apresentação, deve-se criar um novo documento HTML;
- não ser extensível, ou seja, não é possível a especificação e a criação de novos elementos e atributos para situações específicas e, com isso, indicar o significado dos dados;
- ser difícil reutilizar a informação e também alterar ou manter os documentos, pois é necessário alterar o documento como um todo (apresentação e conteúdo), além de faltar suporte à estruturação do documento;

- possuir pouca estrutura semântica, já que seus elementos são agrupados sem seguir uma estrutura pré-definida, tornando-se difícil de encontrar o que se esteja procurando no WWW.

Devido a estas e outras desvantagens, torna-se evidente que a HTML não fornece recursos suficientes para a especificação e o suporte à total variedade de aplicações existentes no ambiente WWW. Mesmo que a HTML atenda à atual demanda dos hipertextos, essa linguagem foi projetada para definir a forma de apresentação de um documento e oferece poucos recursos para a definição semântica do conteúdo de suas informações.

Dessa forma, a HTML não satisfaz a todos os requisitos necessários para atender à nova geração de documentos hipertexto. Já a SGML não possui essas desvantagens, porém é uma linguagem muito complexa. Assim, é necessário obter uma linguagem tão poderosa quanto a SGML e tão simples como a HTML. A partir dessa idéia, um novo padrão foi criado, chamado de XML.

2.4 - XML

A linguagem XML (*Extensible Markup Language*) também foi definida a partir da SGML, sendo um subconjunto da mesma, e foi criada como uma alternativa para a solução dos problemas encontrados em HTML, pois tem uma maior flexibilidade para manipular conteúdos propriamente ditos. O objetivo da XML é fornecer vários benefícios encontrados em SGML e que não estão disponíveis em HTML. Além disso, facilitar o seu aprendizado e a sua utilização se comparada com a complexa linguagem SGML (Mace et al., 1998; McGrath, 1999).

A XML é uma metalinguagem definida como uma forma simplificada da SGML e oferece uma abordagem padrão para descrição, captura, processamento e publicação de informações. A XML não substituirá a HTML, mas irá complementá-la, considerando que a HTML é usada para formatar e mostrar as informações enquanto que a XML representa o significado contextual das mesmas.

Ao contrário da linguagem HTML, a XML não possui tipos de elementos pré-definidos (*tags*). Desse modo, os projetistas podem criar seus próprios elementos de acordo com a aplicação que está sendo projetada. E com esse intuito, a XML dá maior

importância ao conteúdo e à estrutura da informação, sem se preocupar com a forma de sua apresentação. Além disso, a XML foi projetada de modo que qualquer grupo de trabalho pudesse criar sua própria linguagem de marcação, satisfazendo as necessidades específicas de suas aplicações de modo rápido, eficiente e lógico (Connolly et al., 1997). A **Figura 2.3** mostra um documento formatado de acordo com a especificação da XML, encontrada em (Bray et al., 2000). Já em (Connolly, 2000), tem-se diversos artigos relacionados à linguagem XML.

```
<?xml version="1.0" encoding="ISO -8859-1"?>
<!DOCTYPE documento SYSTEM "Documento.dtd">
<?xml-stylesheet type="text/xsl" href="Documento.xsl"?>
<documento>
  <titulo>Servidor de Documentos XML Usando JAVA</titulo>
  <capitulo>
    <titulo>Capítulo 1 - Introdução</titulo>
    <seção>
      <titulo>1.1 - Considerações Iniciais</titulo>
      <conteúdo>
        <parágrafo>...</parágrafo>
        <parágrafo>Com esse crescimento, os documentos eletrônicos estão tornando-se cada
          vez mais complexos, gerando, com isso, uma grande limitação na sua
          manipulação e falta de extensibilidade e intercâmbio com outras aplicações
          (Bosak, 1997). </parágrafo>
        <parágrafo>...</parágrafo>
      </conteúdo>
    </seção>
    <seção>
      <titulo>1.2 - Motivação</titulo>
      <conteúdo>
        <parágrafo>...</parágrafo>
      </conteúdo>
    </seção>
  </capitulo>
</documento>
```

Figura 2.3 – Um documento XML.

Os objetivos propostos para a XML foram de suprema importância no seu desenvolvimento e incluem (Bray et al., 2000; Light, 1999):

- ser usada de forma direta na Internet;
- suportar uma grande variedade de aplicações;
- ser compatível com a SGML;
- ser fácil escrever programas que processam documentos XML;
- o número de recursos opcionais em XML deve ser mantido em um valor mínimo, de preferência zero;

- os documentos XML devem ser legíveis e razoavelmente claros aos usuários;
- o modelo XML deve ser formal e conciso;
- os documentos XML devem ser fáceis de criar;
- a concisão, em marcação XML, deve ser de mínima importância.

A linguagem XML descreve uma classe de objetos de dados (documentos XML) e descreve, parcialmente, o comportamento de programas que os processam (Bray et al., 2000). Esse processamento é realizado através de dois módulos. O primeiro módulo, chamado de processador XML, é usado para ler um documento XML e fornecer acesso ao seu conteúdo e estrutura. A tarefa fundamental do processador XML é interpretar (*parsing*) os dados, ou seja, refere-se ao processo por meio do qual os caracteres que formam o documento XML são categorizados como marcação ou caracteres de dados. O processador envia essas informações para o segundo módulo (aplicação XML), ou seja, ele trabalha em benefício da aplicação, como é mostrado na **Figura 2.4**.

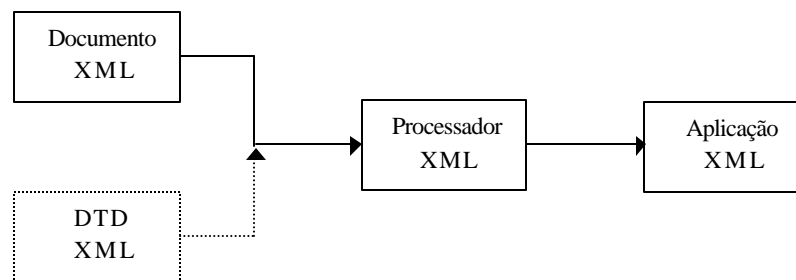


Figura 2.4 – Sistema baseado em XML (McGrath, 1999).

Um documento XML pode ter associado a ele um conjunto de regras que define o que é informação propriamente dita do que é marcação desses dados, descrevendo a estrutura do documento. Essas regras encontram-se em um DTD e o fato de ser mostrado por linhas pontilhadas na **Figura 2.4** indica que seu uso é opcional. A seguir, o conceito de DTD é apresentado.

2.4.1 - DTD

Um DTD (*Document Type Definition*) define regras para a especificação de uma classe de documentos XML, tais como:

- que tipos de elementos podem existir em um documento (um documento sobre um livro, por exemplo, pode conter um título, um índice, capítulo(s), seção(ões), parágrafo(s) entre outros);
- que atributos esses elementos podem ter (o título do livro pode conter um atributo indicando o volume ou a edição);
- como as instâncias desses elementos estão hierarquicamente relacionadas (um capítulo pode conter um título e uma ou mais seções que contêm um ou mais parágrafos).

O DTD permite ao projetista definir a sequência e o aninhamento que os elementos devem obedecer, os valores e os tipos de atributos, e a estrutura que farão parte do documento XML a que ele está associado. Além disso, o projetista pode definir no DTD quais elementos e atributos são permitidos e quais são opcionais, as entidades que devem ou podem aparecer e serem referenciadas, entre outros recursos.

A estrutura especificada em um DTD, segundo sua definição no padrão SGML, possui uma propriedade importante: apenas a estrutura lógica de um documento é descrita, não sendo fornecida nenhuma informação sobre a semântica de apresentação do documento (Brown, 1989). Desse modo, em um livro, por exemplo, é dada a estrutura de seus capítulos, seções e parágrafos, mas nada é definido a respeito do formato de apresentação das informações.

Um DTD é associado a um documento XML através duas formas: incluído no próprio documento XML ou como um documento DTD que é referenciado pelo documento XML. O documento XML da **Figura 2.3** contém uma referência ao DTD “Documento.dtd”, o qual pode ser visto na **Figura 2.5**. Mais informações sobre DTD podem ser obtidas em (Bray et al., 2000).

```

<?xml version="1.0" encoding="ISO -8859-1"?>
<!ELEMENT documento (título, capítulo+)>
<!ELEMENT capítulo (título, seção+)>
<!ELEMENT seção (título, conteúdo)>
<!ELEMENT conteúdo (seção | parágrafo | marcador | figura | tabela | linhaembranco)*>
<!ELEMENT título (#PCDATA | itálico)*>
<!ELEMENT parágrafo (#PCDATA | itálico)*>
<!ELEMENT itálico (#PCDATA)>
<!ELEMENT linhaembranco EMPTY>
<!ELEMENT marcador (mponto | mnumero)*>
<!ELEMENT mponto (#PCDATA | itálico)*>
<!ELEMENT mnumero (#PCDATA | itálico)*>
<!ELEMENT figura (legenda?)>
<!ELEMENT legenda (#PCDATA | itálico)*>
<!ATTLIST figura
    arquivo CDATA #REQUIRED
    nome CDATA #IMPLIED
>
<!ELEMENT tabela (linha+, legenda)>
<!ELEMENT linha (coluna+)>
<!ELEMENT coluna (#PCDATA)>
<!ATTLIST coluna
    posição CDATA #IMPLIED
>

```

Figura 2.5 – Um documento DTD referenciado por um documento XML.

Os documentos XML podem ser bem formados ou válidos. Um documento é bem formado se segue as regras definidas pela especificação XML. Os documentos que não puderem passar por um *parser* XML não são documentos bem formados. Um documento é válido se está de acordo com as restrições indicadas no seu DTD, como mostra a **Figura 2.6**. Se um documento XML for válido, ele também será bem formado. *Parsers* sem validação lêem um arquivo XML e verificam apenas a estrutura sintática do documento XML. *Parsers* com validação verificam se o documento é bem formado e determinam se as *tags* são válidas, se os nomes dos atributos são adequados, se os elementos aninhados estão corretos, etc.

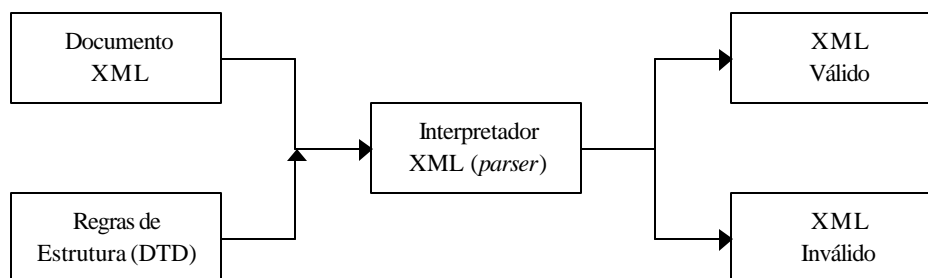


Figura 2.6 – Verificação da estrutura de um documento XML com um parser XML
(McGrath, 1999).

Pode-se notar que o documento XML não trata da apresentação das informações, mas somente do conteúdo a ser apresentado. Sendo assim, existe a necessidade de utilizar outro recurso que seja responsável pelos atributos de apresentação. Esta tarefa pode ser realizada através da utilização de *parsers* específicos ou linguagens apropriadas para associar estilos ao conteúdo de um documento XML.

2.4.2 - Linguagens de Estilos

Como os documentos XML não têm recursos para especificar informações sobre a apresentação de seu conteúdo, eles podem utilizar uma folha de estilos utilizada para formatar sua apresentação. Uma das opções é a utilização dos mecanismos da CSS (*Cascading Style Sheet*) para ajustar fontes, cores, posicionamentos, fundos e muitos outros aspectos de apresentação a esses documentos (Culshaw et al., 1997).

A CSS é uma linguagem declarativa que descreve o relacionamento entre as variáveis em termos de funções e regras de inferência. Esta linguagem especifica, na forma de declarações, como associar estilo a elementos de um DTD como, por exemplo, associar informações de tipo e cor de fonte aos elementos de um documento HTML ou XML. O efeito do estilo afeta o conteúdo do elemento e também de todos aqueles que estão hierarquicamente abaixo dele, a não ser que já se tenha definido outro estilo para alguns desses elementos filhos.

Sua especificação foi desenvolvida ciente do fato de que até mesmo uma linguagem fixa definida por *tags*, como a HTML, pode se beneficiar separando as informações de apresentação das informações de conteúdo propriamente ditas. Além disso, sua especificação pode ser incluída em um documento HTML como sendo parte do documento. Porém, a CSS possui diversas limitações importantes como, por exemplo, ela não pode transformar um arquivo XML em um HTML para apresentação (Culshaw et al., 1997; Johnson, 1999; McGrath, 1999).

A linguagem XSL (*Extensible Stylesheet Language*) foi criada visando definir regras de transformação para serem aplicadas aos elementos XML de modo a oferecer muito mais recursos que a CSS. Como exemplo, a XSL permite a reorganização de conteúdo, o que não é possível com a CSS. Os projetistas da XSL esperam uma coexistência dos dois padrões, sendo a CSS destinada a documentos XML simples e a XSL destinada a documentos complexos que necessitariam da capacidade de

formatação mais poderosa, como acesso à estrutura hierárquica dos elementos (Adler et al., 1997). Mais informações sobre a CSS são encontradas em (Bos, 2000).

A XSL é baseada no padrão DSSSL (*Document Style Semantics and Specification Language*) que é uma linguagem de folha de estilo de padrão internacional para formatação SGML (Adler et al., 1997). Assim como a SGML, a DSSSL é uma linguagem muito poderosa e extensa, o que a torna muito complexa e pouco intuitiva. Apesar de ter incorporado muitas idéias da DSSSL, a XSL é uma linguagem bem mais simples. E qualquer folha de estilo XSL pode ser convertida mecanicamente a uma equivalente em DSSSL.

A finalidade da XSL é fornecer uma folha de estilo poderosa com uma sintaxe de fácil utilidade para expressar como os documentos XML devem ser apresentados. A XSL é poderosa o suficiente para lidar com a eliminação, reorganização e geração de conteúdo juntamente ao processo de formatação. E ainda, pode-se tornar a XSL independente de qualquer tipo de formato resultante.

Dessa forma, através da XSL, um documento XML pode ser transformado em outros formatos de documentos como, por exemplo, RTF, TeX, *PostScript* e HTML, usando ferramentas compatíveis com a XSL, conforme ilustrado na **Figura 2.7**. Além disso, utilizando-se de vários documentos XSL diferentes, o mesmo documento XML pode ser apresentado em diversas formas.

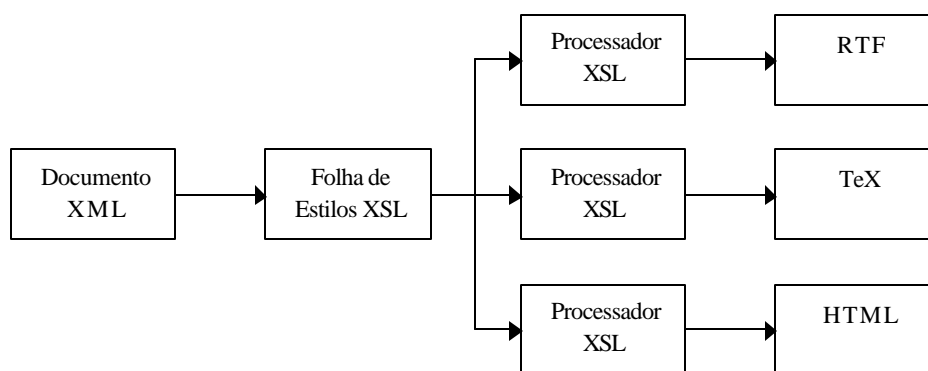


Figura 2.7 – Transformação de um documento XML em outros formatos usando a XSL
(McGrath, 1999).

A **Figura 2.8** mostra um exemplo de um documento formatado de acordo com a especificação da XSL, encontrada em (Adler et al., 1997). Esse documento XSL pode ser aplicado ao documento XML da **Figura 2.3**, cujo DTD é mostrado na **Figura 2.5**, e, através de um processador XSL, gerar o documento HTML da **Figura 2.1**. Esse é um exemplo de conversão de um documento XML para um documento HTML utilizando as especificações de um documento XSL. O documento HTML gerado tem o conteúdo do documento XML e o formato de apresentação definido no documento XSL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
        <TITLE>
          <xsl:value-of select="documento/titulo"/>
        </TITLE>
      </HEAD>
      <BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
        <xsl:apply-templates select="documento"/>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="documento">
    <H1 ALIGN="CENTER">
      <xsl:apply-templates select="titulo"/>
    </H1>
    <BR/>
    <xsl:for-each select="capítulo">
      <BR/>
      <H2 ALIGN="LEFT">
        <xsl:apply-templates select="titulo"/>
      </H2>
      <xsl:for-each select="seção">
        <xsl:if test="titulo">
          <H3 ALIGN="LEFT">
            <xsl:apply-templates select="titulo"/>
          </H3>
          <xsl:apply-templates select="conteúdo"/>
        </xsl:if>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="titulo">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="conteúdo">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="parágrafo">
    <DD>
      <P ALIGN="JUSTIFY">
        <xsl:apply-templates/>
      </P>
    </DD>
  </xsl:template>

  <xsl:template match="text()">
    <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Figura 2.8 – Um documento XSL.

De acordo com (Adler et al., 1997; McGrath, 1999), a XSL tem um conjunto de princípios essenciais do modelo de desenvolvimento que incluem:

- ser usada de forma direta na Internet;
- ser expressa em sintaxe de XML;
- fornecer uma linguagem declarativa para realizar todas as tarefas comuns de formatação;
- fornecer o recurso de se recorrer em uma linguagem de *script* a fim de acomodar tarefas de formatação mais sofisticadas e permitir extensibilidade e integridade;
- ser um sub-sistema da DSSSL;
- ser possível realizar um mapeamento mecânico de uma folha de estilos CSS em uma de XSL;
- deve levar em conta as experiências de uso da linguagem de folha de estilos FOSI (*Format Output Specification Instance*), que também é usada para formatar o modo como documentos devem ser apresentados (FOSI, 1998);
- o número de recursos opcionais em XSL deve ser mantido em um valor mínimo;
- as folhas de estilo XSL devem ser legíveis e razoavelmente claras aos usuários;
- as folhas de estilo XSL devem ser fáceis de ser criadas;
- concisão, em marcação XSL, deve ser de mínima importância.

Um documento XSL pode conter uma série de regras denominadas *templates*. As *templates* são aplicadas a um documento XML, obtendo como resultado o conteúdo do documento XML junto com o formato de apresentação aplicado e organizado de acordo com o especificado no documento XSL.

Sendo assim, a XSL pode fazer o papel de uma linguagem de consulta, pois o conteúdo gerado será apresentado de acordo com os *templates* definidos no documento XSL. Porém, para se tratar especificamente de consultas a documentos XML, foi

definida a linguagem XQL. Em (Froumentin, 2000) tem-se outras informações sobre a linguagem XSL.

2.4.3 - XQL

A linguagem XQL (*XML Query Language*) foi projetada especificamente para documentos XML. É uma linguagem de consulta de propósito geral e fornece uma única sintaxe que pode ser usada para consultas, endereçamento e modelos. A XQL é concisa, simples e poderosa (Robie et al., 1998).

A XQL permite consultar qualquer tipo de dado XML da mesma forma que a linguagem de consulta SQL (*Structured Query Language*) o faz para banco de dados relacionais. O próximo capítulo descreve algumas características da linguagem SQL.

Para saber as características de uma consulta XQL, é útil considerar quatro questões básicas sobre o ambiente em que uma consulta ocorre (Robie, 1999b):

- O que é banco de dados?
- O que é linguagem de consulta?
- Qual é a entrada de uma consulta?
- Qual é o resultado de uma consulta?

A **Tabela 2.1** fornece uma breve resposta para cada uma destas questões, incluindo uma comparação com a SQL, que é amplamente usada para consultas a bancos de dados relacionais:

SQL	XQL
O banco de dados é um conjunto de tabelas.	O banco de dados é um conjunto de documentos XML.
As consultas são feitas em SQL, uma linguagem de consulta que usa tabelas como um modelo básico.	As consultas são feitas em XQL, uma linguagem de consulta que usa uma estrutura XML como um modelo básico.
A cláusula FROM (uma das cláusulas que fazem parte de uma expressão SQL) determina as tabelas que serão examinadas pela consulta.	Uma consulta é determinada por um conjunto de nós de entrada de um ou mais documentos XML.
O resultado de uma consulta é uma tabela contendo um conjunto de linhas.	O resultado de uma consulta é um conjunto de nós de um ou mais documentos XML que podem ser empacotados em um nó raiz, criando um documento XML bem formado.

Tabela 2.1 – Comparação entre SQL e XQL (Robie, 1999b).

A linguagem XQL é uma notação para recuperar informação de um documento XML que pode ser no formato de um conjunto de nós, uma informação sobre relacionamentos de nós, ou pode produzir valores. A especificação não indica o formato de saída. O resultado de uma consulta pode ser um nó, uma lista de nós, um documento XML, um *array* ou uma outra estrutura. Ou seja, a XQL não descreve o formato binário de retorno, mas sim os retornos lógicos. A **Figura 2.9** mostra um exemplo de uma consulta usando a linguagem XQL. Em (Robie, 1999a), tem-se um pequeno tutorial com algumas regras da linguagem.

```
Query:

//título

Result:

<xql:result >
  <título>
    Servidor de Documentos XML Usando JAVA
  </título>
  <título>
    Capítulo 1 - Introdução
  </título>
  <título>
    1.1 - Considerações Iniciais
  </título>
  <título>
    1.2 - Motivação
  </título>
</xql:result>
```

Figura 2.9 – Uma consulta XQL.

A XQL é ainda uma extensão natural à sintaxe padrão da XSL. Ela é uma linguagem de notação para endereçar e filtrar os elementos e texto de documentos XML. Além disso, adiciona as capacidades que XSL fornece para identificar as classes de nós, adicionando lógica *booleana*, filtros, índice em coleções de nós entre outras capacidades. Ela fornece uma notação concisa e compreensível por apontar elementos específicos e por procurar nós com características particulares.

Os principais objetivos de XQL, de acordo com (Robie et al., 1998), são:

- as *strings* XQL devem ser compactas;
- XQL deve ser legível e fácil de escrever;
- a sintaxe XQL deve ser simples para os casos comuns e simples;
- XQL deve ser expressa em *strings* que podem ser encaixadas facilmente em programas, *scripts* e atributos HTML ou XML;
- ser facilmente interpretada (*parsed*);
- ser expressa em *strings* que podem se ajustar naturalmente em URLs (*Uniform Resource Locator*);

- ser capaz de especificar qualquer caminho que possa ocorrer em um documento XML, além de especificar qualquer conjunto de condições para os nós no caminho;
- ser capaz de identificar de modo único qualquer nó em um documento XML;
- as consultas XQL podem retornar qualquer número de resultados, inclusive zero;
- as consultas XQL são declarativas e não procedurais. Elas dizem o que deve ser encontrado e não como. Isto é importante porque uma consulta eficaz deve ser livre para usar índices ou outras estruturas para encontrar resultados eficientemente;
- as condições das consultas XQL podem ser avaliadas em qualquer nível de um documento, e não é esperado navegar a partir da raiz de um documento;
- as consultas XQL retornam resultados em ordem e sem nenhuma repetição de nós.

2.5 - Considerações Finais

Neste capítulo, foram descritas as principais características das linguagens de marcação SGML, HTML e XML, bem como suas vantagens e desvantagens. A XML fornece vários benefícios encontrados em SGML e que não estão disponíveis em HTML, como a separação entre conteúdo e apresentação. Desse modo, é possível alterar o conteúdo de um documento XML sem se preocupar com a forma de sua apresentação.

Foram definidos ainda os conceitos de DTD e de linguagens de estilo. Vale destacar a linguagem de estilo XSL, que é usada para expressar como o conteúdo dos documentos XML deve ser apresentado, além de poder transformá-los em outros formatos de documento.

Além disso, as linguagens de consulta XQL e SQL foram abordadas e comparadas, sendo observado que a XQL permite consultar qualquer tipo de dado XML da mesma forma que a SQL faz para banco de dados relacionais.

O anúncio da Sun Microsystems sobre a criação de um *parser* na linguagem de programação Java para o desenvolvimento de aplicações utilizando a XML e o anúncio da Microsoft sobre o suporte à XML e à XSL na versão 5.0 do Internet Explorer contribuíram significativamente para uma maior utilização dessa tecnologia. A XML tem sido usada pela Microsoft, Netscape, Sun Microsystems, Adobe, IBM entre outras empresas. Como algumas de suas áreas de aplicação pode-se citar *home banking*, tecnologia *push*, automação da Web, integração de banco de dados, distribuição de *software*, publicação científica (linguagem de marcação química) entre outras.

Capítulo 3 - Repositórios de Dados

3.1 - Considerações Iniciais

Nos dias de hoje, existe uma grande quantidade de informações armazenadas em bancos de dados, nos quais podem ser feitos diversos tipos de operações. Governo, bancos, empresas de comércio eletrônico são exemplos de potenciais usuários de banco de dados.

Por outro lado, com a grande difusão de sistemas distribuídos e do modelo cliente/servidor, surgiram vários programas servidores que disponibilizam diversos tipos de informações. Essas informações podem estar armazenadas em diferentes tipos de repositórios.

O capítulo em questão trata sobre banco de dados relacionais e a tecnologia *JavaSpaces*, que são repositórios de dados que trabalham de formas diferentes para realizar o armazenamento e o acesso às informações. Inicialmente é abordada a definição de um sistema de banco de dados, o qual utiliza a linguagem SQL (*Structured Query Language*) para manipular as informações armazenadas. Depois, é a vez de apresentar o funcionamento do serviço *JavaSpaces*, que pode ser usado como um repositório para armazenamento de objetos Java. Além disso, é feita uma comparação entre banco de dados relacionais e *JavaSpaces*, sendo mostradas algumas das principais diferenças entre eles.

3.2 - Sistema de Banco de Dados Relacional

Um sistema de banco de dados é um ambiente de *hardware* e de *software*, composto por: dados armazenados em um banco de dados, um componente central que é o sistema de gerenciamento de banco de dados (SGBD) e os programas de aplicação.

Um SGBD consiste em uma coleção de dados inter-relacionados e um conjunto de programas para acessá-los. Um conjunto de dados, referenciado como banco de dados, contém informações sobre um empreendimento particular. O principal objetivo de um SGBD é prover um ambiente que seja conveniente e eficiente para recuperar e armazenar informações dos bancos de dados (Korth & Silberschatz, 1993).

Além disso, um SGBD é caracterizado pelos seus modelos de banco de dados e pelas suas funções, que implementam um conjunto básico de serviços, os quais definem as capacidades que um sistema deve possuir para ser considerado um sistema de banco de dados. Algumas das principais características de um SGBD são (Melo et al., 1997):

- garantir a consistência do banco de dados e das transações usando estratégias de controle de concorrência;
- garantir a integridade do banco de dados ao final de cada transação;
- dispor de recursos que possibilitem selecionar a autoridade de cada usuário, para ter o controle de seus acessos às informações;
- prover mecanismos de segurança de acesso para consulta ou atualização dos dados;
- dispor as informações em um único local, evitando redundâncias;
- recuperar falhas de *hardware* e *software* através de operações como *backups*.

Para permitir o acesso às informações armazenadas nos bancos de dados, foram criadas linguagens destinadas à sua manipulação. Assim, a linguagem SQL (*Structured Query Language*) foi desenvolvida como forma de interface para o sistema de banco de dados relacional. O objetivo desse sistema é gerar um conjunto de relações que nos permita armazenar informações sem redundância desnecessária e, ainda, poder recuperá-las facilmente.

A linguagem SQL é uma linguagem de controle e acesso a dados. É um dos meios mais utilizados para manipular a informação armazenada em uma base de dados. Por ser uma linguagem não estruturada e de fácil entendimento, a SQL estabeleceu-se como a linguagem padrão para banco de dados relacional e é bastante utilizada nos sistemas de informação das empresas.

A SQL tem a capacidade de gerenciar índices sem a necessidade de controle individualizado de índice corrente. Ela também é capaz de gravar ou cancelar uma série de atualizações feitas à base de dados. Outra característica disponível em SQL é a capacidade de construir visões, que são modos de visualização dos dados na forma de listagens independente das tabelas e da organização lógica dos dados.

Deve-se notar que a linguagem SQL consegue implementar estas soluções pelo fato de estar baseada em bancos de dados, que garantem, por si mesmos, a integridade das relações existentes entre as tabelas e seus índices.

Com o advento da tecnologia de banco de dados em ambiente cliente/servidor, o processamento, manipulação e acesso aos dados passaram a ser feitos no servidor, ficando o lado do cliente apenas com o processamento das aplicações. Com isso, as aplicações tornam-se mais rápidas.

Dentre os SGBDs cliente/servidor encontrados no mercado, os mais conhecidos são: Oracle, Informix, Jasmine, SQLServer, Sybase, DB2 e SQLBase. Os SGBDs são usados principalmente em empresas que possuem aplicações críticas como bancos, indústrias, grandes grupos comerciais entre outras. Mas as médias e pequenas empresas também estão adotando SGBDs devido ao seu baixo custo e à facilidade de implementação. É uma tendência que as aplicações mudem de um ambiente mono-usuário e passem para um ambiente cliente/servidor, devido principalmente à grande popularização da Internet.

3.3 - *JavaSpaces*

As subseções seguintes apresentam a tecnologia *JavaSpaces*. Nelas são abordados a definição da tecnologia, suas relação com a tecnologia Jini, suas operações e o modo como elas trabalham, além das principais vantagens que uma aplicação utilizando esse serviço pode obter.

3.3.1 - Apresentação

Criada para todas as espécies de redes e dispositivos eletrônicos, a tecnologia Jini permite a conexão de qualquer equipamento em qualquer ponto da rede, de modo que eles possam oferecer ou solicitar serviços a um repositório Jini e obter meios de utilizá-los de maneira espontânea (Edwards, 1999). Devido à sua simplicidade, Jini remove as tradicionais barreiras de compatibilidade, confiabilidade e administração, e cria comunidades de equipamentos (também chamadas de federações) sem a intervenção ou conhecimento de algum usuário. Dessa forma, uma máquina fotográfica digital poderia reconhecer uma impressora disponível na rede para imprimir uma imagem colorida em alta-resolução.

Entretanto, em um ambiente distribuído muitas aplicações têm a necessidade de armazenar dados de maneira persistente. O serviço *JavaSpaces* fornece esta capacidade na forma de um serviço da tecnologia Jini (Edwards, 1999). Em *JavaSpaces*, é possível coordenar os participantes em uma federação Jini para criar aplicações distribuídas sofisticadas. Em qualquer caso, a tecnologia *JavaSpaces* pode reduzir significativamente o tempo e o esforço exigidos para projetar e implementar aplicações distribuídas. Além disso, Jini possui mecanismos de tolerância a falhas em sistemas distribuídos, como transações e *leasing*, os quais podem ser utilizados por um serviço *JavaSpaces*. Outras informações sobre a tecnologia Jini podem ser obtidas em (JINI, 2000).

A tecnologia *JavaSpaces* provê um modelo de programação diferente, onde uma aplicação seria como uma coleção de processos cooperando através de um fluxo de objetos entrando e saindo de um ou mais espaços.

Um espaço é um repositório compartilhado onde objetos, chamados de entradas (*entries*), são armazenados. Os processos usam esse repositório como um armazenamento de objetos persistente e mecanismo de troca e, em vez de se comunicar diretamente, eles coordenam suas atividades trocando objetos através desse espaço. Um processo pode escrever (*write*), ler (fazer uma cópia - *read*) ou retirar (*take*) objetos de um espaço. A **Figura 3.1** mostra vários processos interagindo com espaços e usando essas operações. Ao ler ou retirar objetos de um espaço, os processos usam uma comparação simples, baseada nos valores dos campos, para encontrar os objetos que interessam. Se um objeto compatível não for encontrado imediatamente, então um

processo pode ficar esperando até a chegada de um que seja compatível. Em *JavaSpaces*, ao contrário de repositórios convencionais de objetos, os processos não modificam objetos no espaço ou invocam seus métodos diretamente. Para modificar um objeto, um processo tem que explicitamente removê-lo, atualizá-lo e inseri-lo novamente no espaço.

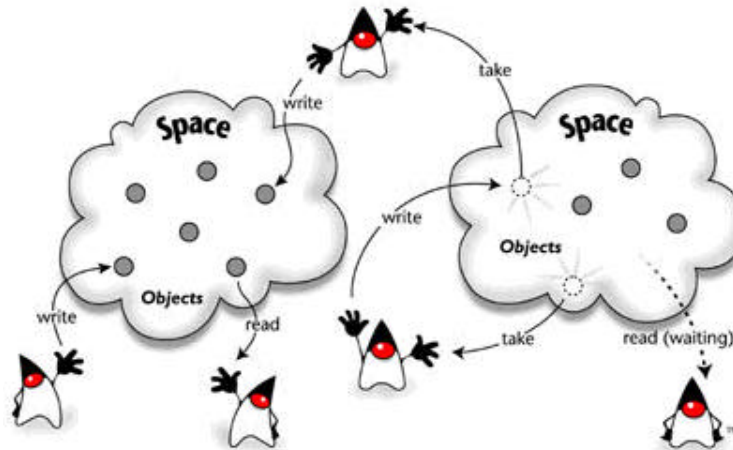


Figura 3.1 – Processos interagindo com espaços diferentes (JAVASPACEs, 2000b).

Além disso, os espaços fornecem armazenamento confiável de objetos de forma persistente. Ao armazenar um objeto em um espaço, ele permanecerá por tempo indeterminado até que seja removido. Porém, é possível determinar um tempo (*lease*) durante o qual um objeto será mantido no espaço. Uma vez armazenado, um objeto permanecerá lá até seu *lease* (que pode ser renovado) terminar ou que um processo o remova explicitamente.

Todas as operações que modificam um serviço *JavaSpaces* são executadas de maneira segura em relação àquele espaço através do uso do serviço de transações Jini, que é um mecanismo de tolerância a falhas, sincronização e controle de concorrência. Dessa forma, se uma operação *write* foi executada com sucesso, quer dizer que aquela entrada foi escrita no espaço, apesar de uma operação *take*, que ocorra inesperadamente, poder removê-la logo a seguir. E se uma operação *take* retornar uma entrada, é porque aquela entrada foi removida do espaço e nenhuma operação futura irá ler ou retirar essa mesma entrada. Em outras palavras, cada entrada no espaço pode ser retirada no

máximo uma vez. Porém, nada impede que duas ou mais entradas em um espaço tenham exatamente o mesmo valor. Transações asseguram que qualquer operação será realizada ou não.

3.3.2 - Operações

Um espaço armazena entradas (*entries*) que são sequências ordenadas de objetos. É possível procurar por entradas em um serviço *JavaSpaces* usando *templates*, que são objetos de entrada que possuem alguns ou todos os seus campos ajustados com os valores específicos da combinação desejada. Os campos restantes agem como coringas e são ajustados com o valor *null*, podendo ser combinados com qualquer valor.

Javaspaces possui um conjunto de operações reduzido. Para manipular qualquer objeto que implementa um serviço *JavaSpaces*, são fornecidas as seguintes operações (Freeman & Hupfer, 1999; JAVASPACEs, 2000a):

- *write*: escreve uma cópia de uma entrada no espaço. Se forem feitas múltiplas chamadas da mesma entrada, então são escritas múltiplas cópias da entrada no espaço.
- *read*: envia uma entrada que é usada como um *template*, retornando uma cópia de um objeto no espaço compatível com esse *template*. Se não houver nenhum objeto compatível no espaço, então *read* pode esperar um certo tempo definido pelo usuário até uma entrada compatível chegar ou até esse tempo se esgotar.
- *take*: funciona como *read*, exceto que o objeto compatível com o *template* é removido do espaço.

Existem ainda as operações *readIfExists* e *takeIfExists*, as quais diferem das operações *read* e *take* pelo fato de não esperarem um objeto chegar no espaço caso não encontre um objeto compatível com o *template*. Além disso, *JavaSpaces* possui outras duas operações utilizadas em situações específicas:

- *notify*: um *template* e um objeto são usados de modo que o espaço notifica a esse objeto sempre que entradas compatíveis com o *template* forem escritas no espaço. É útil em casos onde é necessário interagir com o espaço como em aplicações que utilizam eventos distribuídos.

- *snapshot*: fornece um método para minimizar a serialização que acontece sempre que entradas ou *templates* são enviadas ao espaço. É útil em casos onde é necessário realizar uma operação sobre uma mesma entrada sem necessitar fazer alguma modificação.

3.3.3 - Vantagens de *JavaSpaces*

Uma aplicação modelada utilizando o serviço de *JavaSpaces* pode ter vantagens como (Freeman et al., 1999):

- Ser simples - não é necessário aprender uma interface de programação complexa, mas apenas algumas simples operações.
- Ser poderosa - com apenas algumas operações é possível construir uma grande variedade de aplicações distribuídas sem precisar escrever muito código.
- Suportar protocolos com baixo acoplamento - em *JavaSpaces*, espaços suportam protocolos que são simples, flexíveis e confiáveis, já que não é possível saber quem adicionou ou retirou um objeto do espaço. Isso facilita a criação de diversas aplicações (é possível adicionar componentes de maneira fácil, sem ter que projetar novamente a aplicação inteira), suporta análise global (é possível examinar computação local e coordenação remota separadamente) e aumenta a reutilização de *software* (é possível substituir qualquer componente por outro, contanto que eles obedeçam ao mesmo protocolo).
- Facilitar a escrita de sistemas cliente/servidor - *JavaSpaces* tem diversas funcionalidades acessíveis necessárias em um servidor, como acesso simultâneo por diversos clientes, armazenamento persistente e transações. Em boa parte dos casos é necessário apenas escrever o código cliente e o resto é tratado pelo próprio espaço.

JavaSpaces pode ser facilmente entendida e usada de forma expressiva. Em comparação com outras ferramentas de programação distribuída, a programação com *JavaSpaces*, em muitos casos, facilita o desenvolvimento do projeto, reduz código e

tempo de depuração e viabiliza aplicações que são mais robustas, mais fáceis de fazer manutenção e de integrar com outras aplicações.

3.4 - Banco de Dados Relacionais e *JavaSpaces*

Um serviço *JavaSpaces* pode armazenar dados persistentes que poderão ser recuperados posteriormente. Porém, ele não é um banco de dados relacional ou orientado a objetos. *JavaSpaces* é um serviço projetado para ajudar a resolver problemas em computação distribuída e não para ser usado primariamente como um repositório de dados (embora existam muitos usos de armazenagem de dados para aplicações *JavaSpaces*). Comparando *JavaSpaces* com banco de dados, tem-se que (Freeman et al., 1999, JAVASPACEs, 2000a):

- um banco de dados relacional armazena e manipula os dados diretamente através de uma linguagem de consulta. Entradas em *JavaSpaces* só podem ser recuperadas pelo tipo e forma serializada de cada campo, já que não possui uma linguagem geral para intermediar uma consulta. Sendo assim, não é possível retornar um conjunto de entradas;
- um banco de dados orientado a objetos fornece uma imagem orientada a objetos do dado armazenado que pode ser modificado e utilizado. *JavaSpaces* não provê um mecanismo de persistência transparente, ou seja, dados não podem ser modificados dentro do espaço, de modo que todas as entradas são cópias dos objetos originais.

Estas diferenças existem porque *JavaSpaces* é projetado para um propósito diferente de um banco de dados relacional ou orientado a objetos. Pode-se dizer que *JavaSpaces* é algo entre um sistema de arquivos e um banco de dados. Um serviço *JavaSpaces* pode ser usado para armazenar dados persistentes, como armazenar dados preferenciais dos usuários, os quais podem ser recuperados posteriormente através do nome ou um identificador do usuário.

Mais importante ainda, *JavaSpaces* pode armazenar mais do que apenas dados. Isto significa que qualquer programa, dispositivo ou informação que seja baseado em objetos pode se juntar a um sistema *JavaSpaces*. Dessa forma, pacotes de dados são

tratados exatamente como qualquer outro objeto mantido no espaço. Esta habilidade aumenta muito a capacidade de coordenar diferentes funções e processos em uma rede.

3.5 - Considerações Finais

Neste capítulo, discutiu-se sobre duas formas de repositórios de dados que trabalham de modos diferentes para realizar o armazenamento e o acesso às informações desejadas, sendo apresentado o funcionamento de cada um, além de uma comparação entre esses repositórios, que são banco de dados relacionais e *JavaSpaces*.

O serviço *JavaSpaces*, que é um repositório para armazenamento de objetos Java, é projetado para ajudar a resolver problemas em computação distribuída e tem um propósito diferente de um banco de dados relacional ou orientado a objetos. Porém, existem diversas aplicações que utilizam *JavaSpaces* como um repositório de dados e, neste trabalho, ele é utilizado para o armazenamento persistente de documentos XML.

A escolha pela tecnologia *JavaSpaces* para o armazenamento de documentos XML deve-se ao fato de *JavaSpaces* ser uma tecnologia simples e fácil de usar. Além disso, *JavaSpaces* é assunto de outros projetos em desenvolvimento do grupo de trabalho, o que possibilita que esse projeto seja utilizado e integrado mais facilmente aos demais.

Capítulo 4 - Acesso e Manipulação de Dados XML

4.1 - Considerações Iniciais

A informação armazenada em documentos XML deve, em algum momento, ser lida por um programa para executar uma determinada função, como visualizar, modificar ou imprimir. A linguagem de programação Java, neste trabalho, entra como um fator de essencial importância no que diz respeito à conversão, por exemplo, de um documento XML em um documento HTML. Através da implementação de um *parser* nesta linguagem, é possível recuperar os elementos dos arquivos DTD e XML, aplicar as estruturas e os estilos de acordo com a XSL e a CSS, gerando finalmente um documento HTML, que contém o resultado que pode ser apresentado no *browser*.

Porém, qualquer outra linguagem de programação (não necessariamente Java) pode ser utilizada para implementação de um *parser* XML. Isto é possível devido, em grande parte, à existência de duas APIs (*Application Programming Interface*): SAX (*Simple API for XML*) e DOM (*Document Object Model*). Elas foram criadas com o propósito de permitir aos programadores acessar a informação armazenada em documentos XML sem precisar escrever um *parser* em uma linguagem de programação específica. Isto é possível, pois essas duas APIs têm implementações disponíveis para diversas linguagens. Assim, se a informação do documento for mantida no formato XML versão 1.0 e se uma dessas duas APIs for utilizada, a aplicação pode usar qualquer *parser* XML disponível (Idris, 1999).

O assunto abordado neste capítulo refere-se às formas de acesso e manipulação de dados XML. Primeiramente, é apresentada a linguagem de programação Java, utilizada para a implementação deste trabalho. Em seguida, são discutidos os prós e os contras de se utilizar as APIs SAX e DOM, além do porquê de ter-se optado por DOM, mostrando suas principais interfaces. Por fim, são descritas as bibliotecas utilizadas para a realização das manipulações aos dados XML: Xerces, Xalan e GMD-IPSI XQL.

4.2 - A Linguagem de Programação Java

A linguagem de programação Java foi desenvolvida no início da década de 90 pela Sun Microsystems (Van Hoff, 1998). Inicialmente, o objetivo de Java não era ser utilizada na Internet, mas sim criar uma linguagem independente de plataforma para ser usada em dispositivos eletrônicos, como fornos de micro-ondas e controles remotos. Por esse motivo, o propósito principal de Java era ser independente de plataforma, o que levou os desenvolvedores da Internet a buscar em Java soluções para os seus problemas. Assim, a linguagem passou a ser utilizada na Internet em grande escala.

Java ganhou popularidade rapidamente e, em 1995, a Netscape e a Microsoft a incorporaram em seus *browsers*, o que impulsionou grandemente a difusão da linguagem. O sucesso de Java não se deu por ela ser poderosa ou sofisticada, mas porque o conjunto de suas características a torna simples, concisa e bem definida.

Em termos da Internet, a linguagem Java tem se destacado como uma das principais linguagens de programação e, por isso, foi escolhida e utilizada para o desenvolvimento deste trabalho.

Apesar do uso de Java na Internet ter sido direcionado inicialmente à implementação de *applets*, que possibilitavam interações em documentos HTML para sua apresentação no ambiente WWW, deve-se considerar ainda que tal linguagem possui um vasto conjunto de recursos para construção de aplicações *stand-alone* (independentes de documentos HTML e do ambiente WWW).

A linguagem Java favorece o desenvolvimento de aplicações distribuídas por estar disponível para diversas plataformas e por ter a capacidade de movimentar código pela rede. Dessa forma, Java permite escrever componentes portáteis que podem ser

distribuídos no ambiente WWW, permitindo ainda que as aplicações desenvolvidas para esse ambiente possam trocar código entre elas.

Java é uma linguagem que tem um grande conjunto de qualidades de modo que pode ser utilizada em diversas aplicações. Dentre elas, pode-se citar: orientação a objeto, interligação com sistemas CORBA (*JavaIDL*), suporte ao desenvolvimento de servidores (*servlets*), gerenciamento (JMAPI), internacionalização, independência de plataforma (oferecendo suporte desde *smartcards* até *mainframes*), acesso a base de dados (JDBC), objetos distribuídos (*JavaRMI*), comércio eletrônico (*JavaCommerce*), componentização (*JavaBeans*), multimídia (*JavaMedia*), telefonia (*JavaTelephony*), suporte à voz (*JavaSpeech*), segurança (*JavaSecurity*) entre outras. E a lista de capacidade para os desenvolvedores só tende a crescer porque toda a indústria de informática suporta e colabora para que a plataforma Java venha a atender a todos os tipos de aplicações.

Fazendo um resumo de suas principais características, tem-se que Java é uma linguagem simples, poderosa, segura, orientada a objeto, distribuída, robusta, interpretada (ou compilada dinamicamente), independente de plataforma, *multithreaded*, com um mecanismo eficiente de coleta de lixo e manipulação de exceções (JAVA, 2000a; Naughton, 1996).

4.3 - SAX

A API SAX (*Simple API for XML*) foi criada para permitir o acesso à informação armazenada em documentos XML em qualquer linguagem de programação (Megginson, 1998). Ela provê esse acesso não como uma árvore de nós, mas como uma sequência ordenada de eventos. A aplicação interpreta esses eventos de uma maneira significativa e cria o seu próprio modelo de objetos baseada nestes eventos. Assim, SAX torna-se mais rápida e simples, porém as aplicações que usam essa interface precisam das seguintes condições (Idris, 1999):

- criação de seu próprio modelo de objetos personalizado;
- criação de uma classe que capture os eventos SAX (gerados pelo *parser* SAX conforme ele obtém o documento XML) e crie adequadamente o modelo de objetos.

Uma vantagem de SAX é que o documento inteiro não fica residente na memória, o que é interessante em situações onde há processamento de documentos gigantes. Os *parsers* SAX percorrem um documento XML ativando métodos que tratam de cada caso, como, por exemplo, início, conteúdo e fim de elemento. A **Figura 4.1** mostra a representação do documento XML da **Figura 2.2** como uma sequência ordenada de eventos.

```
start document
start element: documento
start element: título
characters: Servidor de Documentos XML Usando Java
end element: título
start element: capítulo
start element: título
characters: Capítulo 1 - Introdução
end element: título
start element: seção
start element: título
characters: 1.1 - Considerações Iniciais
end element: título
start element: conteúdo
start element: parágrafo
characters: ...
end element: parágrafo
start element: parágrafo
characters: Com esse crescimento ... (Bosak, 1997).
end element: parágrafo
start element: parágrafo
characters: ...
end element: parágrafo
end element: conteúdo
end element: seção
start element: seção
start element: título
characters: 1.2 - Motivação
end element: título
start element: conteúdo
start element: parágrafo
characters: ...
end element: parágrafo
end element: conteúdo
end element: seção
end element: capítulo
end element: documento
end document
```

Figura 4.1 – Representação do documento XML da Figura 2.2 através de SAX.

SAX é mais adequada para propósitos onde é preciso ler um documento XML completo do início ao fim e executar alguma tarefa, tal como construir uma estrutura de

dados que represente um documento ou resumir a informação contida em um documento (como, por exemplo, calcular a média de um certo elemento). SAX não é muito útil em casos onde se deseja modificar a estrutura do documento de alguma forma complicada que envolva troca de elementos que estão aninhados. De qualquer forma, SAX poderia ser utilizado quando simplesmente deseja-se trocar atributos ou conteúdos de elementos. Como exemplo, não é possível reordenar os capítulos de um livro usando SAX, porém é permitido trocar os conteúdos de qualquer elemento com um determinado atributo por um outro valor qualquer.

Desse modo, SAX é recomendada em casos onde sua informação é estruturada de tal maneira que seja mais fácil criar seu próprio mapeamento do que representar a informação como uma árvore (Idris, 1999).

4.4 - DOM

O poder de XML está na sua capacidade de representar a estrutura da informação baseada em como cada parte dessa informação se relaciona com as demais (Bray et al., 2000). Os documentos estruturados têm a propriedade de poder ser aninhados um dentro do outro, apresentando uma estrutura em forma de árvore.

DOM (*Document Object Model*) fornece uma interface independente de plataforma e linguagem para a estrutura e o conteúdo de documentos HTML e XML. Ele descreve uma linguagem neutra capaz de representar qualquer documento HTML ou XML bem formado em forma de uma árvore e tratar a informação armazenada nesses documentos como um modelo de objetos hierárquicos (Idris, 1999). DOM cria uma árvore de nós, baseada na estrutura e na informação do documento, sendo que o acesso à informação pode ser feito através de interações com essa árvore. A **Figura 4.2** mostra a representação do documento XML da **Figura 2.2** em forma de nós em uma estrutura de árvore.

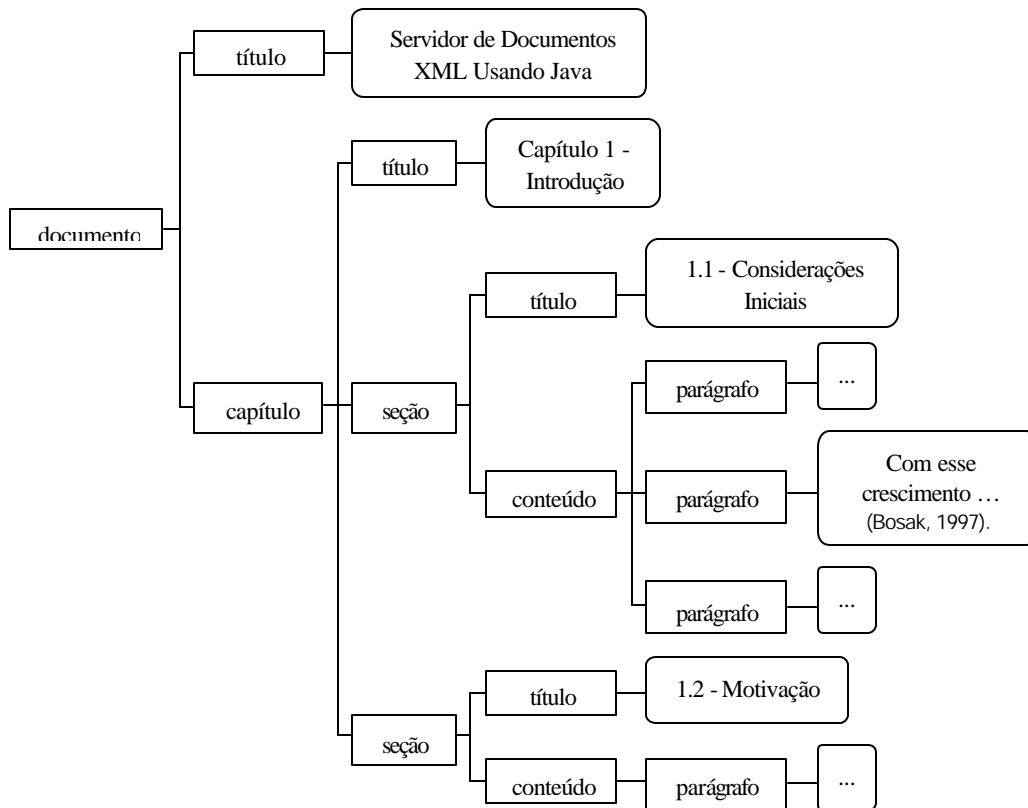


Figura 4.2 – Representação do documento XML da Figura 2.2 através de DOM.

DOM busca fornecer um modelo padrão único na forma como são organizados os diversos objetos que formam os documentos. Ele também busca padronizar uma interface desses objetos para facilitar na navegação em documentos e no processamento deles. Além disso, DOM preserva a sequência dos elementos lidos a partir dos documentos HTML e XML, como se fosse um documento.

Os principais objetivos de projeto de DOM são (McGrath, 1999):

- fornecer um conjunto de objetos e interfaces suficientes para representar o conteúdo e a estrutura dos documentos HTML e XML sem perda de informações significativas;
- realizar isso de uma forma independente de plataforma e linguagem;
- fornecer funcionalidade de criação de objeto poderosa o suficiente para permitir que documentos HTML e XML sejam criados completamente a partir do zero;

- fornecer uma base sólida e extensível na qual podem ser adicionadas camadas DOM no futuro.

Diferente de SAX, em DOM o *parser* faz quase tudo: lê o documento, cria um modelo de objeto em uma linguagem (neste trabalho, em Java) e fornece uma referência a este modelo de objeto para poder controlá-lo (Idris, 1999).

Com o uso de DOM, os usuários se beneficiam de uma interface homogênea tanto para a HTML como para a XML. Eles também podem mover suas aplicações para qualquer plataforma compatível com DOM sem ter que efetuar qualquer alteração de código. E ainda é possível fazer a transformação de um documento XML para HTML, por exemplo, utilizando as especificações XSL.

4.5 - Bibliotecas

A conversão de um documento XML em uma estrutura de dados em árvore ou em uma sequência ordenada de eventos pode ser realizada através da linguagem Java. Neste caso, SAX e DOM são recursos que podem ser utilizados, pois existem bibliotecas que utilizam essas APIs em suas implementações e que disponibilizam vários métodos para realizar diversas manipulações sobre os documentos XML. Assim, essas APIs auxiliam o programador a manipular e a apresentar documentos XML em suas aplicações.

Porém, para realizar as manipulações definidas neste trabalho sobre os documentos XML, a API DOM pode ser utilizada de modo mais abrangente do que a API SAX, pois SAX não é muito adequado em atividades mais elaboradas onde é necessário modificar a estrutura do documento de forma mais complicada. Já DOM possui diversos métodos para realizar manipulações sobre a estrutura e o conteúdo do documento, permitindo que programas possam adicionar, modificar ou apagar nós da árvore de objetos a ele associados.

Existe, em Java, uma série de interfaces necessárias que estão definidas de acordo com a recomendação DOM, cada uma representando um determinado objeto da árvore de nós. Dentre estas interfaces, pode-se citar:

- **Node** - representa qualquer objeto em uma árvore de nós. Possui várias interfaces herdeiras para determinar se o objeto é um elemento, atributo,

texto, etc. Nela estão definidos os principais métodos que fornecem informações comuns a todos os objetos como nome, valor e tipo;

- `Document` - herda da interface `Node` e representa o documento XML como uma árvore de nós, fornecendo acesso ao nó raiz da árvore gerada. Isso significa que ele possui apenas um nó filho, que é o nó raiz;
- `Element` - herda da interface `Node` e representa um elemento em uma árvore de nós;
- `Attr` - herda da interface `Node` e representa um atributo contido na interface `Element`;
- `Text` - herda da interface `Node` e representa o texto contido em um `Element` ou em um `Attr`;
- `DocumentType` - herda da interface `Node` e representa o DTD associado à árvore de nós;
- `NodeList` - representa um conjunto de nós. Como exemplo, pode conter quais são todos os filhos de um determinado nó.

As interfaces possuem diversos métodos que relacionam uma interface com a outra, podendo obter diversos tipos de informações sobre cada objeto da árvore. Em (Apparao et al., 1998), podem ser encontradas mais informações sobre todas as interfaces, assim como uma descrição de cada método que cada interface oferece.

A seguir, são apresentadas as bibliotecas Xerces, Xalan e GMD-IPSI XQL, escritas na linguagem Java e usadas no desenvolvimento deste trabalho. Essas bibliotecas utilizam as interfaces DOM em suas implementações para aplicar suas manipulações sobre os documentos XML. Aqui são descritas sua definição e suas principais classes e interfaces. No próximo capítulo, é demonstrado como cada uma é utilizada para a realização do trabalho.

4.5.1 - Xerces

Xerces é um *parser* XML, escrito em Java, que contém classes e métodos utilizados para interpretar, validar e manipular documentos XML (XERCES, 2000). O pacote implementa as APIs DOM – *Level 1* (Apparao et al., 1998) e *Level 2* versão 1.0 (Le

Hors et al., 2000) – e SAX – versão 1 (Megginson, 1998) e versão 2 (Megginson, 2000) – e também suporta a implementação de XML *Schema* – versão 1.0 (Connolly & Thompson, 2000).

O *parser* Xerces tem várias utilidades em diversas aplicações e pode ser usado, por exemplo, para construir servidores WWW compatíveis com a linguagem XML, assegurar a integridade de dados *e-business* escritos em XML e, ainda, ser usado para a próxima geração de aplicações que utilizarão XML como o formato de dados.

Dentre as diversas classes existentes no pacote Xerces, existe uma, chamada `DOMParser`, que disponibiliza o *parser* para converter um documento XML em um objeto definido de acordo com a interface `Document`. Assim, o documento XML estará representado como um objeto DOM, em forma de uma árvore de nós.

Além disso, é possível determinar diversas características que podem ser analisadas quando se estiver executando o *parser* como, por exemplo, validar o documento XML de acordo com o seu DTD, caso o documento XML tenha um DTD associado. Outras informações sobre o pacote Xerces, além de sua documentação, estão disponíveis em (XERCES, 2000).

4.5.2 - Xalan

A biblioteca Xalan é um processador XSL, escrito em Java, que permite converter documentos XML em HTML, texto ou em outros documentos XML (XALAN, 2000). Para realizar a transformação de um documento XML em um outro documento, de acordo com o definido pela especificação contida em um documento XSL, é necessário a utilização dos elementos:

- `XSLTProcessorFactory` - classe que cria um processador utilizado para realizar transformações;
- `XSLTProcessor` - interface, para um objeto criado por uma `XSLTProcessorFactory`, que é o processador utilizado para realizar as transformações;

- `XSLTInputSource` - classe que representa um documento XML ou XSL para ser utilizada como parâmetro de entrada pelo `XSLTProcessor` na realização das transformações;
- `XSLTResultTarget` - classe que contém o resultado de uma transformação realizada.

Dessa forma, a biblioteca possui recursos suficientes que permitem converter um documento XML em um outro documento. Como exemplo, é possível passar como parâmetros um documento XML e um XSL representados como uma árvore de nós, através da classe `XSLTInputSource`, retornando uma nova árvore de nós que representa um documento HTML, através da classe `XSLTResultTarget`. Para obter outras informações sobre a biblioteca Xalan, além de sua completa documentação, pode-se consultar em (XALAN, 2000).

4.5.3 - GMD-IPSI XQL

A ferramenta GMD-IPSI XQL é uma aplicação de consulta e armazenamento, escrita em Java, para ser aplicada em grandes documentos XML (GMD-IPSI, 2000). Ela foi criada baseada em duas tecnologias que são: uma implementação persistente de objetos DOM e uma implementação completa da linguagem XQL de acordo com a sintaxe definida em (Robie et al., 1998).

Esta ferramenta possui diversas classes para realizar consultas, através da XQL, e retornar qualquer tipo de dado XML. Dentre suas classes, destacam-se as mais relevantes para o desenvolvimento do trabalho:

- `XQL` - oferece métodos para configurar e executar consultas XQL que são aplicadas em qualquer nó de uma árvore DOM;
- `XQLResult` - contém o resultado de uma consulta realizada, que pode ser representado como uma árvore de nós;
- `XMLWriter` - pode ser usado para escrever um objeto DOM ou parte dele em um `Writer` (classe para escrever um *stream* de caracteres).

Em (GMD-IPSI, 2000), tem-se disponível uma completa documentação, assim como podem ser encontradas mais informações sobre a ferramenta.

4.6 - Considerações Finais

Como foi visto neste capítulo, a linguagem Java pode ser utilizada para implementação de *parsers* que interpretam um documento XML e geram, por exemplo, um documento HTML para apresentação em um *browser*. Java também possui outras características necessárias na implementação deste trabalho como portabilidade, ser distribuída e ser direcionada para a Internet, permitindo movimentar código pela rede.

Foram abordados, ainda, os padrões de API, SAX e DOM, discutindo-se de que maneira cada um realiza o acesso aos dados de documentos XML. SAX é recomendada quando aplicações têm que criar seus próprios modelos de objeto personalizados. Ao contrário, DOM já cria seu modelo de objetos como uma árvore de nós, baseado na estrutura e informação contidas no documento XML.

Porém, SAX não é adequado em casos onde se deseja realizar manipulações mais elaboradas, como as definidas para a realização deste trabalho. Já DOM possui diversos métodos que permitem adicionar, modificar ou apagar nós da árvore de objetos de forma mais abrangente.

Além disso, foram discutidas as principais interfaces DOM escritas na linguagem Java, que disponibilizam vários métodos para realizar a manipulação sobre os dados XML, além das bibliotecas utilizadas no desenvolvimento do trabalho. Essas bibliotecas estão escritas em Java e utilizam as interfaces DOM em suas implementações. As bibliotecas descritas foram: Xerces (um *parser* XML escrito em Java), Xalan (um processador XSL que permite converter documentos XML em outros formatos) e GMD-IPSI XQL (uma ferramenta de consulta a dados XML).

Capítulo 5 - O Servidor XML

5.1 - Considerações Iniciais

Esse capítulo refere-se à implementação do protótipo do servidor XML. Inicialmente, são vistas a definição do que o servidor é capaz de realizar e quais as tecnologias utilizadas para o seu desenvolvimento. Em seguida, são descritos os métodos definidos em sua interface, criada para a realização dos serviços solicitados pelas aplicações clientes, além de como esses métodos funcionam. Também são apresentados uma aplicação gráfica, criada para ilustração do uso de alguns desses métodos, e os testes realizados no servidor XML.

O detalhamento do desenvolvimento do servidor descrito a seguir está baseado na descrição da sua funcionalidade e dos recursos utilizados, considerando-se a base teórica obtida nos capítulos anteriores.

5.2 - Definição e Tecnologias Utilizadas

O servidor XML trabalha sobre um repositório de documentos XML e XSL (que também são documentos XML) e permite realizar diversas manipulações sobre esses documentos, tais como adicionar, consultar e apagar. Ele é capaz de receber consultas nas linguagens XQL ou XSL, retornando conteúdos em várias formas, tais como:

- um documento XML do repositório;
- parte de um documento XML, tornando-se um novo documento XML;

- um novo documento em uma outra linguagem diferente de XML, se a consulta for feita pela linguagem de transformação XSL. Assim, a linguagem desse novo documento é dependente da transformação feita pela XSL;
- um novo documento XML gerado através da XSL.

Como já foi visto anteriormente, o modelo DOM é capaz de representar qualquer documento XML bem formado em forma de uma árvore de nós e tratar a informação armazenada nesses documentos como um modelo de objetos hierárquicos. Tem-se ainda que o serviço *JavaSpaces* é um repositório compartilhado onde objetos Java são armazenados persistentemente. *JavaSpaces* é utilizado, nesse projeto, para o armazenamento dos documentos XML na forma de objetos DOM, que é a forma na qual são realizadas as manipulações sobre esses documentos. Isto é possível, pois a linguagem Java possui uma série de interfaces definidas de acordo com a recomendação DOM. Sendo assim, como um objeto DOM é um objeto Java, que pode ser armazenado no serviço *JavaSpaces*, este serviço pode armazenar todos os documentos XML que serão enviados ao servidor XML.

Além disso, a árvore de nós gerada por DOM pode ser transformada em qualquer outro formato de documento, desde que tais formatos sejam conhecidos. Desse modo, é possível, por exemplo, fazer a transformação de um documento XML para HTML utilizando uma especificação XSL.

Os objetos DOM são adicionados dentro de uma classe, chamada *Name*, que são as entradas (*entries*) do espaço *JavaSpaces*. Essa classe possui um campo *name* de valor único usado para a identificação das árvores DOM que são adicionadas no espaço, pois em *JavaSpaces* é permitido que duas ou mais entradas sejam armazenadas contendo o mesmo valor. A classe possui, ainda, um campo contendo a árvore DOM e outro com o conteúdo do documento DTD associado ao documento XML (contido na árvore DOM do *Name*), caso o documento tenha um DTD, ou *null*, em caso contrário. Esse campo é utilizado para verificar se o objeto DOM armazenado no espaço é válido, ou seja, se ele está de acordo com as regras definidas no seu DTD. Instâncias da classe *Name* podem ser criadas, apagadas, ter seu campo *name* alterado e ainda é possível obter quais *names* identificam os objetos DOM que retornam algum conteúdo em uma determinada consulta XQL.

O servidor permite ainda incluir novos documentos XML no repositório *JavaSpaces* de duas formas: como um documento a mais no repositório ou dentro de outro documento XML já existente. Outra função disponível no servidor é que um documento XML pode ser apagado por inteiro ou por partes. Para isso, a parte do documento que será apagada é declarada através de uma *query* definida na linguagem XQL.

Para o desenvolvimento deste trabalho, foram utilizadas bibliotecas que disponibilizam os serviços necessários e que realizam as funções que podem ser executadas pelo servidor XML. As bibliotecas e suas respectivas versões são:

- Xerces - versão 1.2.2, disponível em (XERCES, 2000);
- Xalan - versão 1.2.2, disponível em (XALAN, 2000);
- GMD-IPSI XQL - versão 1.0.2, disponível em (GMD-IPSI, 2000);
- *JavaSpaces* - versão 1.0.1, disponível em (JAVASPACEs, 2000b);
- Jini - versão 1.0.1, disponível em (JINI, 2001);
- Java - versão 1.3.1, disponível em (JAVA, 2000b).

Para se executar uma aplicação que utilize um serviço *JavaSpaces*, é necessário ter a biblioteca Jini instalada para disponibilizar esse serviço e que os seguintes serviços estejam rodando:

- um Servidor HTTP: usado para transferência de código entre clientes *JavaSpaces*;
- um RMI *Activation Daemon*: usado para controlar os estados dos serviços, desativando ou reativando serviços, caso estejam sendo utilizados ou não;
- um Serviço de *Lookup*: que permite a clientes procurar e encontrar serviços Jini (como *JavaSpaces*) que estão disponíveis na rede local;
- um *Transaction Manager*: necessário se as aplicações *JavaSpaces* fizerem uso de transações (não é utilizado neste trabalho);
- e finalmente um Serviço *JavaSpaces*.

Os serviços acima foram configurados de acordo com as informações obtidas em (Hupfer, 2000). Nessas configurações, são determinadas, por exemplo, se o serviço *JavaSpaces* é persistente ou transiente e o nome dado ao espaço *JavaSpaces*.

5.3 - A Interface do Servidor XML

Para o desenvolvimento do servidor XML, foi construída uma interface Java implementada por um *proxy*, escrito em Java, através do qual é feita a conexão entre as aplicações cliente e o servidor XML. A **Figura 5.1** mostra uma visão geral da arquitetura gerada para a utilização do servidor XML.

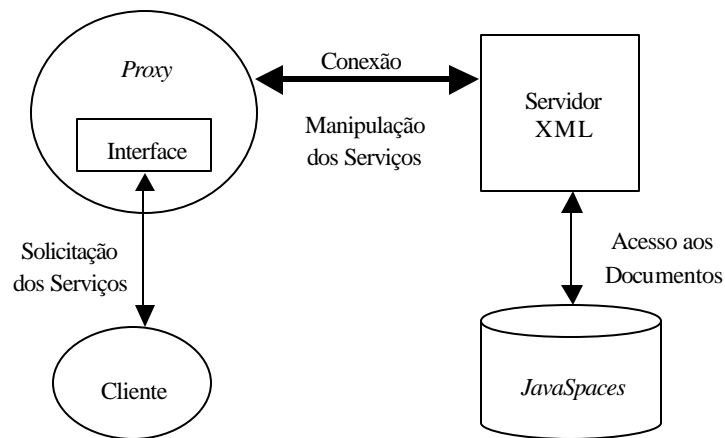
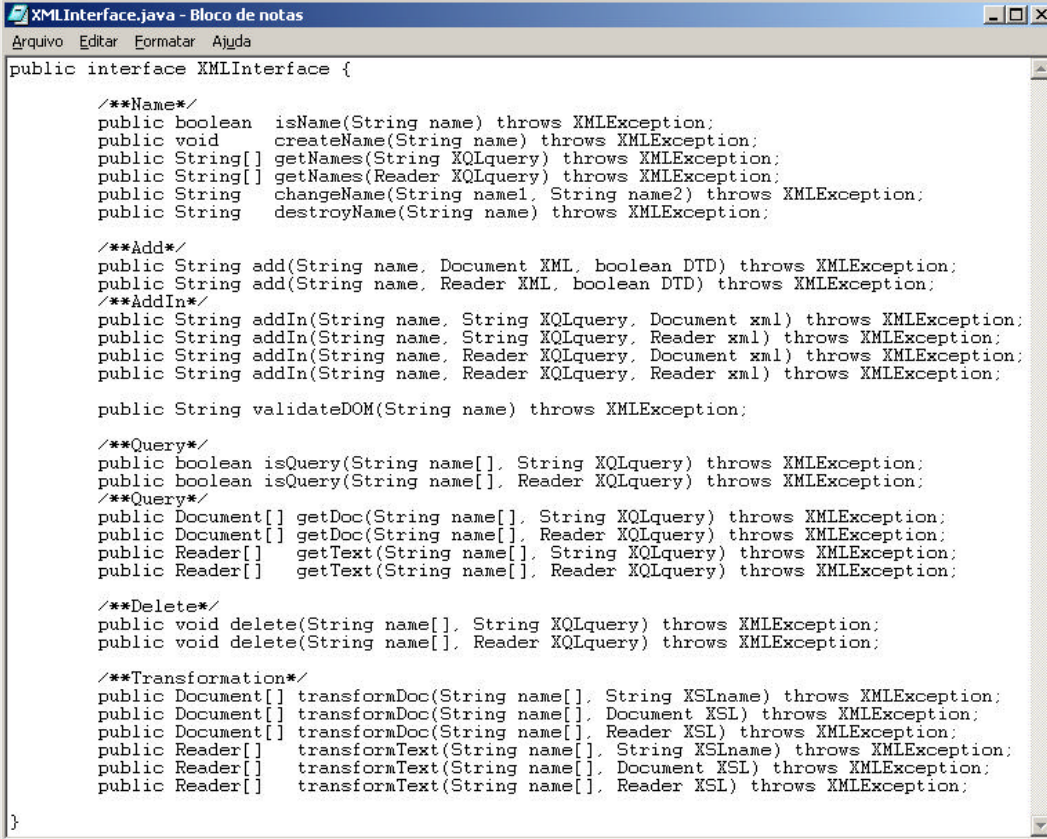


Figura 5.1 – Arquitetura do Servidor XML.

Nessa interface, estão definidos os métodos que são executados pelo servidor XML, sobre os documentos armazenados no espaço *JavaSpaces*, para a realização dos serviços solicitados pelas aplicações clientes. A seguir, tem-se a descrição de cada método contido nessa interface, que pode ser vista na **Figura 5.2**.

- *isName*: verifica se um *name* pertence ao espaço *JavaSpaces*;
- *createName*: cria uma classe *Name* com o valor do campo *name* dado (que deve ser único);



```

XMLInterface.java - Bloco de notas
Arquivo  Editar  Formatar  Ajuda

public interface XMLInterface {

    /**Name*/
    public boolean  isName(String name) throws XMLException;
    public void     createName(String name) throws XMLException;
    public String[] getNames(String XQLQuery) throws XMLException;
    public String[] getNames(Reader XQLQuery) throws XMLException;
    public String   changeName(String name1, String name2) throws XMLException;
    public String   destroyName(String name) throws XMLException;

    /**Add*/
    public String add(String name, Document XML, boolean DTD) throws XMLException;
    public String add(String name, Reader XML, boolean DTD) throws XMLException;
    /**AddIn*/
    public String addIn(String name, String XQLQuery, Document xml) throws XMLException;
    public String addIn(String name, String XQLQuery, Reader xml) throws XMLException;
    public String addIn(String name, Reader XQLQuery, Document xml) throws XMLException;
    public String addIn(String name, Reader XQLQuery, Reader xml) throws XMLException;

    public String validateDOM(String name) throws XMLException;

    /**Query*/
    public boolean isQuery(String name[], String XQLQuery) throws XMLException;
    public boolean isQuery(String name[], Reader XQLQuery) throws XMLException;
    /**Query*/
    public Document[] getDoc(String name[], String XQLQuery) throws XMLException;
    public Document[] getDoc(String name[], Reader XQLQuery) throws XMLException;
    public Reader[]   getText(String name[], String XQLQuery) throws XMLException;
    public Reader[]   getText(String name[], Reader XQLQuery) throws XMLException;

    /**Delete*/
    public void delete(String name[], String XQLQuery) throws XMLException;
    public void delete(String name[], Reader XQLQuery) throws XMLException;

    /**Transformation*/
    public Document[] transformDoc(String name[], String XSLName) throws XMLException;
    public Document[] transformDoc(String name[], Document XSL) throws XMLException;
    public Document[] transformDoc(String name[], Reader XSL) throws XMLException;
    public Reader[]   transformText(String name[], String XSLName) throws XMLException;
    public Reader[]   transformText(String name[], Document XSL) throws XMLException;
    public Reader[]   transformText(String name[], Reader XSL) throws XMLException;

}

```

Figura 5.2 – Interface do Servidor XML.

- *getNames*: retorna um *array* contendo os campos *names* que identificam quais objetos DOM retornam algum conteúdo de acordo com a *query*. Se a *query* for definida como *null*, o *array* retorna todos os *names* que estão no espaço;
- *changeName*: troca o valor do campo *name*;
- *destroyName*: apaga *Name* do espaço, identificada pelo seu campo *name*;
- *add*: adiciona um novo documento XML no espaço, na forma de objeto DOM, somente se ele for válido;
- *addIn*: adiciona um novo documento XML dentro de um outro documento XML já existente dentro do espaço. A *query* indica em que ponto dentro do documento existente o novo documento é adicionado;
- *validateDOM*: verifica se o objeto DOM, identificado pelo seu *name* e que possua um DTD, é válido;

- *isQuery*: verifica se ao menos um dos objetos DOM, identificados pelos seus *names*, retorna algum conteúdo em relação à *query*;
- *getDoc*: aplica a *query* sobre os objetos DOM, identificados pelos seus *names*, retornando um objeto DOM para cada *name*;
- *getText*: aplica a *query* sobre os objetos DOM, identificados pelos seus *names*, retornando um *Reader* (classe para ler um *stream* de caracteres) para cada *name*;
- *delete*: apaga o resultado da *query* aplicada sobre os objetos DOM, identificados pelos seus *names*;
- *transformDoc*: realiza a transformação do documento XSL sobre os objetos DOM, identificados pelos seus *names*, retornando um objeto DOM para cada *name*;
- *transformText*: realiza a transformação do documento XSL sobre os objetos DOM, identificados pelos seus *names*, retornando um *Reader* para cada *name*;

5.4 - Funcionamento dos Métodos da Interface

Antes de realizar qualquer manipulação com os dados dos documentos XML, deve-se, primeiramente, adicioná-los no espaço *JavaSpaces*.

Adicionando um Novo Documento

Quando o método *add* é solicitado, uma instância da classe *Name* é criada, contendo o campo *name* com o valor dado e o documento XML na forma de um objeto DOM. Caso esse *Name* já exista, mas se o seu campo contendo o objeto DOM for igual a *null*, o documento pode ser armazenado normalmente nesse *Name*. Isso acontece quando um *Name* é criado através do método *createName*, pois, nesse caso, o campo contendo o objeto DOM é iniciado com valor igual a *null*.

Caso o documento XML possua um DTD, através da interface *DocumentType*, é possível obter o nome de seu arquivo DTD. Esse arquivo então é lido e seu conteúdo é

armazenado no campo contendo o documento DTD da classe *Name*. Caso o documento não tenha um DTD, esse campo terá valor igual a *null*.

Para realizar o *parsing* sobre um documento XML definido como um objeto da classe *Reader*, visto nos métodos *add* e *addIn* da **Figura 5.2**, é necessário utilizar a biblioteca Xerces. Assim é possível obter uma árvore DOM do documento XML com uma interface *Document* e verificar se o documento é válido. Agora, se o documento já estiver definido de acordo com a interface *Document* (já existe uma árvore DOM para ele), o seu DTD pode ser obtido, caso exista, porém não é verificada sua validade, pois essa condição é verificada somente na hora de se realizar o *parsing*.

Dessa forma, se o documento XML possuir um DTD, ele é adicionado no repositório, como um objeto DOM, somente se for válido. Porém, o usuário poderá definir que um documento XML não possui um DTD associado a ele (mesmo que exista um), podendo, então, adicioná-lo no repositório sem verificar sua validade. Nesse caso, o campo contendo o conteúdo do DTD da classe *Name* fica com valor *null*.

Caso o documento XML possua o DTD incluído no próprio documento, ele também é adicionado no repositório somente se for válido, porém o campo contendo o conteúdo do DTD é guardado como *null*, sendo esta uma limitação da biblioteca usada.

Adicionando um Novo Documento em um já Existente

Através do método *addIn*, é possível adicionar um novo documento dentro de um outro já existente no espaço. Para ilustrar o seu funcionamento, a **Figura 5.3** mostra a representação DOM de um documento XML que esteja armazenado no espaço, cujo conteúdo é um documento que possui um único capítulo.

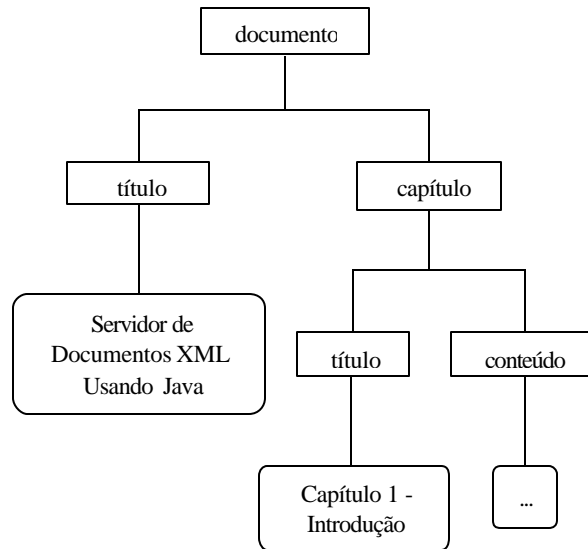


Figura 5.3 – Ilustração da representação DOM de um documento XML armazenado no espaço.

Deseja-se, agora, inserir um novo documento XML que contém um novo capítulo a ser adicionado dentro desse documento XML já existente no espaço. A representação DOM desse novo documento pode ser vista através da **Figura 5.4**.

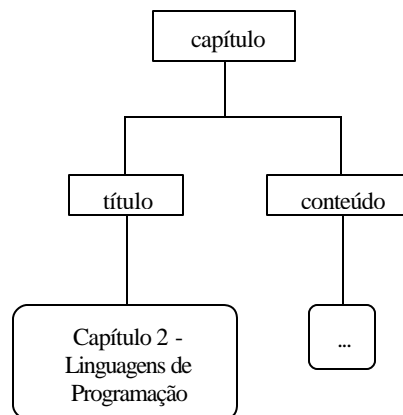


Figura 5.4 – Ilustração da representação DOM do novo documento XML.

Para a realização do método *addIn*, é dada uma *query* (nesse caso, `/documento`) que indica onde esse novo documento XML será adicionado no documento XML existente no repositório. Após ser realizada a operação, um único documento será formado, cuja representação DOM pode ser observada na **Figura 5.5**.

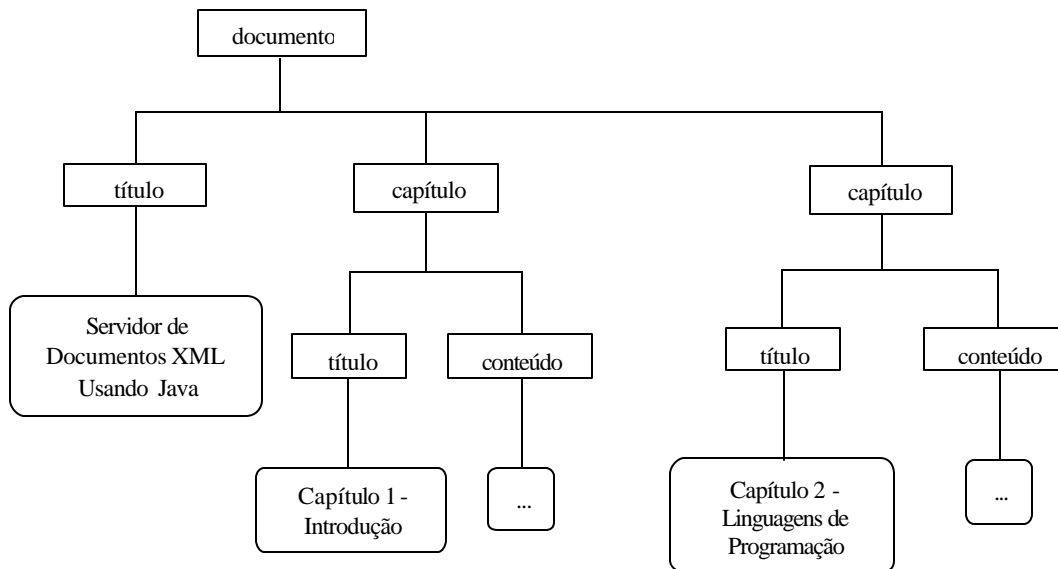


Figura 5.5 – Representação DOM do documento após a operação addIn.

O novo documento XML é adicionado normalmente ao documento existente no repositório sem verificar sua validade antes e depois da operação. Porém, é possível saber se, após ser executada a operação, o novo documento é válido, através do método *validateDOM*. Para verificar se esse DOM é válido, são criados dois arquivos temporários: um arquivo XML contendo a representação XML do objeto DOM e um arquivo DTD contendo o seu DTD, cujo conteúdo está contido no campo que contém o conteúdo do seu DTD da classe *Name*. Depois de se criar os arquivos temporários, é realizado o *parser* do arquivo XML criado, através do uso da biblioteca Xerces, sendo, então, verificado se ele está de acordo ou não com as regras definidas no seu DTD.

Para o seu correto funcionamento, o método *addIn* requer que a *query* dada retorne como resultado um único objeto do tipo *Element*, retornando um erro caso não seja atendida essa condição.

Apagando Dados XML

Para apagar dados de documentos XML, é utilizado o método *delete*. Como exemplo do funcionamento desse método, é dada uma *query*, `//capítulo/título`, que será aplicada ao objeto DOM da **Figura 5.3**. Após ser realizada a operação, o documento terá a representação DOM mostrada na **Figura 5.6**.

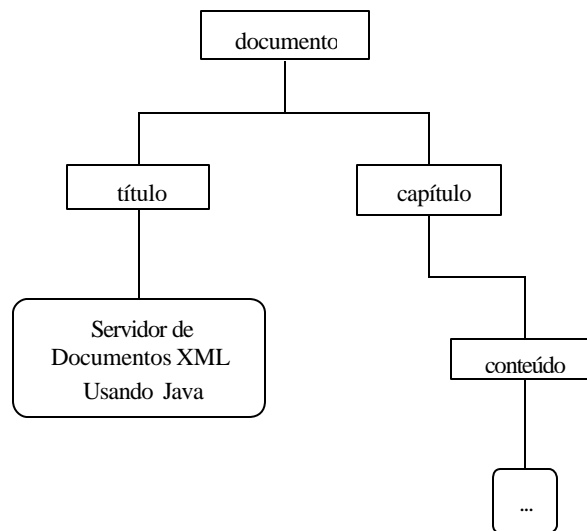


Figura 5.6 – Representação DOM do documento após a operação delete.

Da mesma forma que o método *addIn*, o método *delete* requer que a *query* dada retorne como resultado um objeto do tipo *Element*, porém permite que seja retornado um ou vários objetos desse tipo.

Operações de Transformação

Nos métodos *transformDoc* e *transformText*, a biblioteca Xalan é utilizada para transformar um documento XML em um outro documento, de acordo com o definido pela especificação contida em um documento XSL, que é utilizado nessa transformação. Uma das classes dessa biblioteca, *XSLTInputSource*, permite representar um documento XML ou XSL que esteja definido como uma interface *Document* (árvore DOM) ou em modo texto, usando a classe *Reader*. Já a classe *XSLTResultTarget* permite retornar o resultado na forma de uma interface *Document* (árvore DOM), assim como em texto usando um objeto *Writer* (classe para escrever um *stream* de caracteres).

Conforme definido nos métodos *transformDoc* e *transformText* da interface, o documento XSL pode ser passado sob três formatos: como um objeto *Document*, um *Reader* ou através de uma *String*. Caso ele seja passado como uma *String*, isso

indica que o documento XSL faz parte do repositório, sendo que essa *String* indica o campo *name* que identifica o objeto DOM desse documento XSL no repositório.

Operações de Consulta

Através da linguagem XQL, é possível realizar consultas a qualquer tipo de dado XML. Para ilustrar o funcionamento das operações de consulta, será aplicada uma *query*, `//título`, ao objeto DOM da **Figura 5.3**. A **Figura 5.7** mostra a representação DOM do documento retornado após ser realizada a operação. Observa-se que nesse objeto DOM é criado uma *tag root* com o valor `xql:result`, para garantir que o objeto DOM sempre será bem formado.

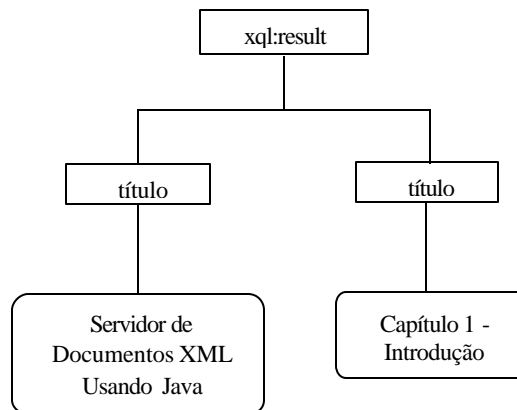


Figura 5.7 – Representação DOM do documento após a operação de consulta.

Para realizar as operações de consulta existentes nos métodos da interface, é utilizada a biblioteca GMD-IPSI XQL. A classe `XQL`, definida nessa biblioteca, oferece métodos para configurar e executar consultas definidas na linguagem XQL, que podem ser aplicadas em qualquer nó de uma árvore DOM. O resultado de uma consulta pode ser obtido através da classe `XQLResult`, a qual pode ser representada como uma árvore de nós ou utilizar a classe `XMLWriter` para escrever a consulta em um `Writer`.

Como já foi visto, *JavaSpaces* utiliza *templates* para procurar por entradas (objetos) armazenadas em seu espaço, sendo que o resultado retornado é um objeto compatível com esse *template*. Dessa forma, não há uma maneira de saber, de uma só

vez, quais são todas as entradas que estão armazenadas no espaço, pois, para um dado *template*, pode ser retornado um mesmo objeto várias vezes, sem que todos os objetos do espaço sejam retornados. Sendo assim, *names* que forem sendo criados são armazenados em uma estrutura *Vector* para que seja possível identificar quais *names* estão armazenados no espaço. Isso é utilizado, como exemplo, no método *getNames*, onde é necessário que se conheça previamente todos os *names* contidos no espaço.

Servidor XML e Proxies

Não fica claro ao usuário que operações são realizadas nos *proxies* ou no servidor XML. É justamente essa a função do *proxy* na arquitetura Jini, esconder do usuário o que é feito localmente do que é feito no servidor. Isso permite que a interface entre o *proxy* e o servidor não precise ser pública podendo ser livremente modificada para permitir a migração de funcionalidade entre os lados cliente (*proxies*) e servidor em futuras versões do *software*.

Por exemplo, pode ser observado, que existem alguns métodos da interface que retornam objetos da classe *Reader*. No entanto, existem os métodos definidos em classes como *XSLTResultTarget* e *XMLWriter*, utilizadas para a realização de algumas das manipulações, que retornam objetos derivados da classe *Writer*. Acontece que esses métodos são executados no lado do servidor, sendo ainda necessário passar o resultado para o lado do cliente. Então, no *proxy*, onde é feita a leitura dos dados retornados pelo servidor, esses dados são retornados como objetos da classe *Reader*, ficando, dessa forma, compatíveis com o resultado retornado pelos métodos definidos na interface.

5.5 - Um Cliente Gráfico

Para demonstrar a funcionalidade de alguns métodos definidos na interface do servidor XML, foi construída uma aplicação cliente. Essa aplicação possui interfaces gráficas que ilustram a conexão com o servidor e como podem ser realizadas algumas das manipulações permitidas.

A aplicação cliente inicia apresentando a tela mostrada na **Figura 5.8**. Nessa tela, o usuário entra com os dados necessários para realizar a conexão com o servidor

XML. Tais dados são: o *host* do servidor, sua porta de conexão e o espaço *JavaSpaces* onde estão armazenados os documentos XML que lhe interessam.

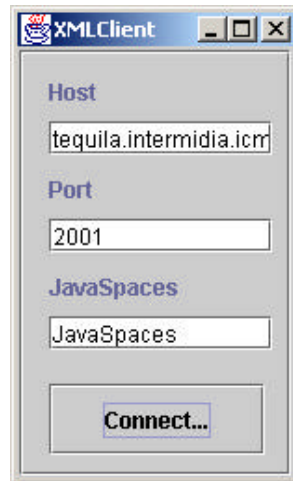


Figura 5.8 – Conexão com o servidor XML.

Após conseguir conectar ao servidor, uma segunda tela é apresentada, como mostra a **Figura 5.9**. Essa tela apresenta as funções disponíveis para a realização de manipulações sobre os documentos XML como: adicionar, apagar, consultar e transformar. Além disso, é possível realizar manipulações com *names* que são utilizados para identificar os objetos DOM armazenados no repositório.

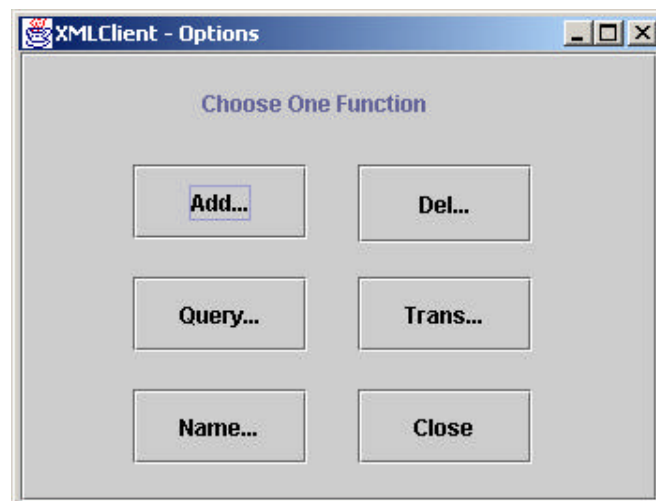


Figura 5.9 – Funções contidas na aplicação cliente.

A primeira função disponível na aplicação serve para adicionar documentos no repositório. Na tela que é apresentada, o usuário indica o nome do arquivo XML ou XSL a ser adicionado no repositório, se este arquivo possui ou não um DTD e um *name* para sua identificação dentro do repositório. Além disso, ele escolhe de que forma deseja que esse arquivo seja adicionado no repositório: como um novo documento ou incluído em um documento já existente. Caso escolha a segunda opção, o usuário deve indicar, através de uma *query*, em que ponto do documento existente ele deseja adicionar o novo arquivo. Essa *query* pode ser simplesmente digitada ou pode estar contida em um arquivo, de modo a exemplificar uma *query* no formato Reader. O usuário pode ainda definir se deseja verificar se o documento continua sendo válido de acordo com o seu DTD após a operação ser realizada, caso esse documento possua um DTD. A **Figura 5.10** mostra um exemplo do processo de adicionar um documento no repositório.




Figura 5.10 – Processo para adicionar um documento.

A próxima função serve para apagar dados de um documento armazenado no repositório. Na tela que aparece, o usuário seleciona o documento, identificando pelo seu *name*, no qual será efetuada a operação. Ele deve indicar, através de uma *query*, qual parte do documento deseja-se apagar. Nessa operação, há ainda uma opção de *Preview* para que o usuário possa ver quais são os dados que serão apagados de acordo com a *query* indicada. Essa opção de *Preview* é um exemplo da utilização do método *getDoc* definido na interface do servidor, pois retorna o resultado no formato Document. A **Figura 5.11** mostra um exemplo do processo de apagar dados de um documento do repositório, sendo utilizado a opção de *Preview* para mostrar os dados que serão apagados.

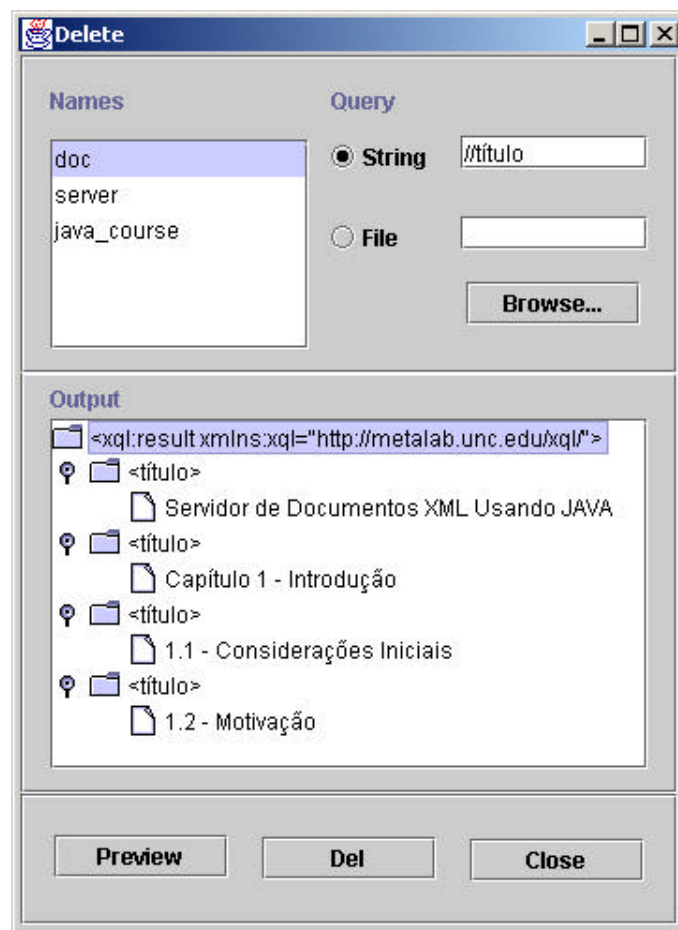


Figura 5.11 – Processo para apagar dados de um documento.

Outra função disponível serve para a realização de consultas aos documentos. Nessa função, o usuário indica o nome do documento, através de seu *name*, e a *query* a ser aplicada. Da mesma forma, existe a opção de *Preview* para que o usuário possa ver os dados retornados de acordo com a *query* dada. O usuário pode ainda salvar a consulta realizada em um arquivo, para realizar manualmente alguma manipulação com esses dados. A **Figura 5.12** mostra um exemplo do processo de consulta aos dados de um documento do repositório, sendo utilizado a opção de *Preview*.

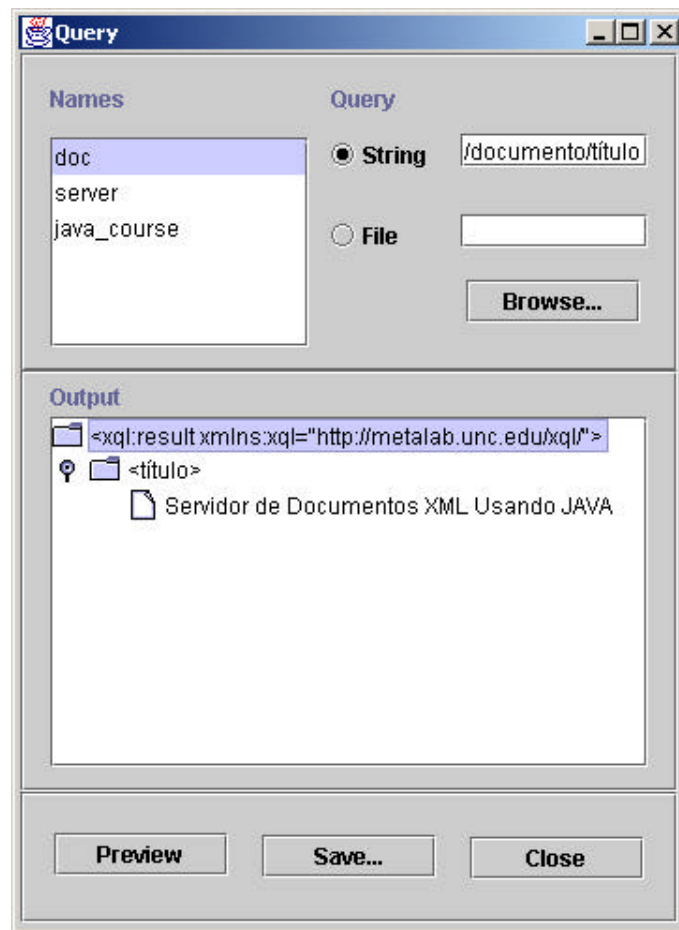


Figura 5.12 – Processo para consultar dados de um documento.

Não muito diferente é a implementação da função de transformação. Nessa tela, o usuário indica o nome do documento XML, através de seu *name*, e o nome do documento XSL, que pode estar contido em um arquivo ou fazer parte do repositório, sendo, nesse caso, indicado pelo seu *name*. A opção de *Preview* apresenta o resultado da

transformação realizada, que pode ser salvo em um arquivo. A **Figura 5.13** mostra um exemplo do processo de transformação de um documento do repositório, utilizando a opção de *Preview*.

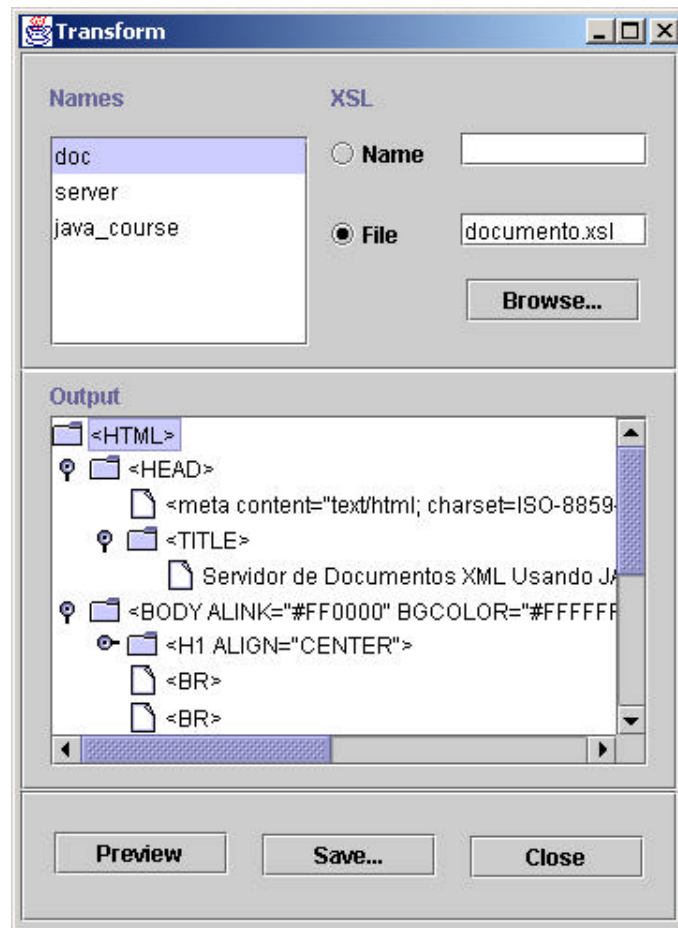


Figura 5.13 – Processo para transformação de um documento.

Por fim, é possível realizar manipulações com *names* que são utilizados para identificar os objetos DOM armazenados no repositório. Nessa função, o usuário pode criar, alterar ou destruir um *name* do repositório (nesse caso, significa que o documento é apagado do repositório). Para isso, o usuário escolhe uma função a ser aplicada sobre o *name* digitado. Caso escolha a função para alterar *name*, o usuário precisa selecionar um *name* da lista de *names*, o qual terá seu campo alterado. A **Figura 5.14** mostra um exemplo do processo de manipulação de *names*.

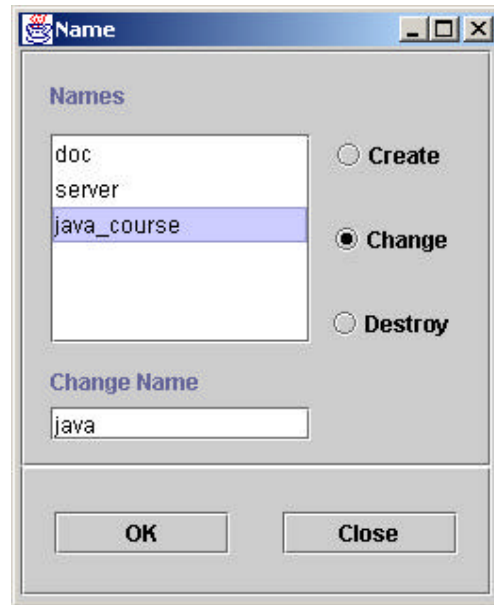


Figura 5.14 – Processo para manipulação com *names*.

5.6 - Testes Realizados

Além dessa aplicação gráfica, também foi construída uma outra aplicação cliente, que não utiliza interfaces gráficas, para realizar testes em todos os métodos disponíveis na interface do servidor XML. Essa aplicação foi construída somente para a realização de testes básicos, de modo a garantir o funcionamento básico do servidor.

Nessa aplicação, foram criadas funções que utilizam todos os métodos definidos na interface, inclusive os que não estão disponíveis na aplicação gráfica, como *isName* e *isQuery*. Dentre os testes realizados, incluem-se:

- o funcionamento de cada método da interface foi testado ao menos uma vez;
- foram conectados 3 clientes ao servidor, solicitando as mesmas manipulações e ao mesmo tempo;
- em relação aos métodos que permitem realizar a mesma manipulação com mais de um documento em uma única solicitação, como *delete*, *getDoc* e *transformDoc*, foram realizados testes sobre 3 documentos do repositório, de modo que foram retornados 3 objetos de acordo com o resultado desejado. Esse teste não foi realizado na aplicação gráfica, pois nela só é permitido realizar manipulações com um documento de cada vez;

- foram criadas situações em que a operação solicitada não retornava resultados válidos.

Durante todo o processo de realização dos testes, os resultados foram obtidos conforme o esperado ou foi retornada uma mensagem de erro adequada, em caso de operações inválidas. Pode-se dizer então que o protótipo do servidor possui uma funcionalidade mínima, pois o servidor atendeu a todas as solicitações que lhe foram feitas, obtendo-se sempre o resultado esperado. O servidor não parou de funcionar em nenhum dos testes efetuados.

Apesar de não se ter realizado testes exaustivos e mais rigorosos, os testes apresentados são considerados como satisfatórios, para garantir o *status* de aplicação em estado *beta* para o servidor XML.

5.7 - Considerações Finais

O servidor XML descrito neste capítulo permite realizar diversas manipulações sobre um repositório de documentos XML. Dentre essas manipulações, o servidor é capaz de adicionar ou apagar documentos no repositório, além de realizar consultas sobre os mesmos. Também é possível retornar um novo documento XML ou em uma outra linguagem, desde que a consulta seja feita por uma linguagem de transformação, como é o caso da linguagem XSL.

O serviço *JavaSpaces*, que é um repositório de objetos Java, é utilizado neste trabalho para o armazenamento de documentos XML, os quais são armazenados na forma de objetos DOM, pois Java possui uma série de interfaces definidas de acordo com a recomendação DOM.

Além disso, o servidor possui uma interface implementada por um *proxy*, em Java, o qual realiza a conexão entre as aplicações cliente e o servidor XML. Nessa interface, estão descritos os métodos que definem as manipulações que o servidor é capaz de executar.

Capítulo 6 - Conclusões

6.1 - Considerações Iniciais

Hoje em dia, a linguagem XML vem sendo muito utilizada em diversas aplicações tanto no meio acadêmico quanto no meio comercial. Porém, a disponibilidade de uma grande quantidade de dados XML apresenta várias questões que o padrão XML não discute, por exemplo, como os dados XML podem ser extraídos de grandes documentos XML.

Este capítulo tem por objetivo apresentar as conclusões a respeito do desenvolvimento deste trabalho, bem como as contribuições obtidas e as sugestões para a realização de trabalhos futuros, dando continuação ao seu desenvolvimento.

6.2 - Resultados e Contribuições

Apesar de existir uma linguagem projetada especificamente para consultar qualquer tipo de dado XML, a linguagem XQL, e outra específica para fazer transformações em documentos XML, a linguagem XSL, existe dificuldade em se conseguir implementações abertas de servidores de dados capazes de usar essas linguagens. Por isso, neste trabalho foi construído um servidor de documentos XML capaz de realizar diversas operações sobre documentos XML e XSL armazenados nele, usando as linguagens XQL e XSL para esse fim. Dentre essas operações, o servidor é capaz de:

- incluir novos documentos XML no repositório de duas formas: como um documento a mais no repositório ou dentro de um documento XML já existente;
- receber consultas nas linguagens XQL ou XSL, retornando conteúdos em diversos formatos;
- apagar um documento XML por inteiro ou somente parte dele.

Dessa forma, o servidor facilita diversos tipos de transformações que podem ser aplicadas a documentos XML. Como exemplo, para a edição de um documento XML, ao invés de consultar e retornar o documento por inteiro é possível retornar somente a parte do documento necessária para a edição, sendo possível ainda que essa parte (uma vez editada) seja adicionada de volta ao seu local de origem.

O servidor também permite modificar o modo como documentos XML são apresentados através do uso da linguagem XSL. Através do uso de vários documentos XSL diferentes, o mesmo documento XML pode ser apresentado sob diversas formas.

Para o desenvolvimento deste trabalho, foram utilizadas e integradas diversas tecnologias, cada uma desempenhando sua função para a construção do servidor XML.

6.3 - Sugestões para Trabalhos Futuros

Por se tratar do desenvolvimento de um protótipo de *software*, que serviu basicamente para garantir o funcionamento dos métodos definidos na interface do servidor XML, a atual implementação pode ser melhorada em vários aspectos em futuras versões.

Uma das considerações que podem ser feitas é o desenvolvimento de uma interface gráfica que disponibilize os métodos da interface do servidor XML de uma maneira que mais amigável ao usuário dentro de ferramentas de desenvolvimento como o VisualAge da IBM.

Alguns dos métodos da interface do servidor podem ser implementados de forma mais eficiente, sem prejuízo de compatibilidade com aplicações clientes já escritas. Um exemplo desse caso é o método *validateDOM*. O modo como foi implementado requer que sejam criados arquivos temporários para verificar se um objeto DOM está de acordo ou não com as regras definidas no seu DTD, caso ele tenha

um DTD associado. Esse método poderia ser implementado de uma outra maneira que não necessitasse criar os arquivos temporários. Da mesma forma, pode-se criar uma maneira de não só obter os DTDs que estão em um arquivo separado de seus documentos XML, mas também aqueles que estão incluídos em documentos XML.

Neste trabalho, o serviço *JavaSpaces* é utilizado para o armazenamento dos documentos XML na forma de objetos DOM. Um serviço que *JavaSpaces* disponibiliza é o uso de transações, que são mecanismos de tolerância a falhas, sincronização e controle de concorrência. O trabalho foi desenvolvido sem utilizar transações e em uma versão futura poderia ser incluído o uso desse serviço.

Finalmente, devido a importância que documentos XML provavelmente terão no futuro do WWW, um servidor de documentos XML deverá ser tão eficiente quanto os bancos de dados relacionais são hoje, para isso novos métodos de armazenamento e manipulação de documentos XML (que dispensem o uso do *JavaSpaces*) precisam ser estudados em projetos conjuntos com grupos especializados em Banco de Dados.

Referências Bibliográficas

(Adler et al., 1997) Adler, S.; Berglund, A.; Clark, J.; Cseri, I.; Grosso, P.; Marsh, J.; Nicol, G.; Paoli, J.; Schach, D.; Thompson, H. S.; Wilson, C. *A Proposal for XSL*. Agosto 1997. <http://www.w3.org/TR/NOTE-XSL.html>

(Apparao et al., 1998) Apparao V.; Byrne, S.; Champion, M.; Isaacs, S.; Jacobs, I.; Le Hors, A.; Nicol, G.; Robie, J.; Sutor, R.; Wilson, C.; Wood, L. *Document Object Model (DOM) Level 1 Specification*. Outubro 1998. <http://www.w3.org/TR/REC-DOM-Level1/>

(Bos, 2000) Bos, B. *Cascading Style Sheets*. Fevereiro 2000.
<http://www.w3.org/Style/CSS/>

(Bosak, 1997) Bosak, J. *XML, Java, and the Future of the Web*. XML: Principles, Tools and Techniques. World Wide Web Journal, v. 2, n. 4, pp. 219-227, 1997.

(Bray et al., 2000) Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E. *Extensible Markup Language (XML) 1.0 (Second Edition)*. Outubro 2000.
<http://www.w3.org/TR/REC-xml>

- (Brown, 89) Brown, H. *Standards for Structured Documents*. The Computer Journal, v. 32, n. 6, pp. 505-514, 1989.
- (Cleveland, 1998) Cleveland, G. *SGML: An Overview and Criteria for Use*. Julho 1998.
<http://ifla.inist.fr/VI/5/op/udtop9/udtop9.htm>
- (Connolly et al., 1997) Connolly, D.; Khare, R.; Rifkin, A. *The Evolution of Web Documents: The Ascent of XML*. XML: Principles, Tools and Techniques. World Wide Web Journal, v. 2, n. 4, pp. 119-128, 1997.
- (Connolly, 2000) Connolly, D. *Extensible Markup Language (XML)*. Fevereiro 2000.
<http://www.w3.org/XML/>
- (Connolly & Thompson, 2000) Connolly, D.; Thompson, H. *XML Schema*. Abril 2000.
<http://www.w3.org/XML/Schema.html>
- (Culshaw et al., 1997) Culshaw, S.; Leventhal, M.; Maloney, M. *XML and CSS*. XML: Principles, Tools and Techniques. World Wide Web Journal, v. 2, n. 4, pp. 109-118, 1997.
- (Deutsch et al., 1998) Deutsch, A.; Fernandez, M.; Florescu, D.; Levy, A.; Suciu, D. *XML-QL: A Query Language for XML*. Agosto 1998. <http://www.w3.org/TR/NOTE-xml-ql/>
- (Edwards, 1999) Edwards, W. K. *Core JINI*. Prentice Hall PTR, 1999.
- (FOSI, 1998) *Adept 8.0 FOSI Conversion Support*. Novembro 1998.
http://www.arbortext.com/Customr_Support/Updates_and_Technical_Notes/Adept_8_Updates/FOSI_Conversion/body_fosi_conversion.html

(Freeman & Hupfer, 1999) Freeman, E.; Hupfer, S. *Make Room for JavaSpaces*.
Novembro 1999. http://www.javaworld.com/javaworld/jw-11-1999/jw-11-jiniology_p.html

(Freeman et al., 1999) Freeman, E.; Hupfer, S.; Arnold, K. *JavaSpaces Principles, Patterns, and Practice*. Addison Wesley, 1999.

(Froumentin, 2000) Froumentin, M. *Extensible Stylesheet Language (XSL)*. Fevereiro 2000. <http://www.w3.org/Style/XSL/>

(GMD-IPSI, 2000) *GMD-IPSI XQL Engine: Version 1.0.2*. Novembro 2000.
<http://xml.darmstadt.gmd.de/xql/>

(Hupfer, 2000) Hupfer, S. *The Nuts and Bolts of Compiling and Running JavaSpacesTM Programs*. Janeiro 2000.
<http://developer.java.sun.com/developer/technicalArticles/jini/javaspaces/>

(Idris, 1999) Idris, N. *Should I use SAX or DOM?* Maio 1999.
<http://developerlife.com/saxvsdom/default.htm>

(ISO, 1986) *International Standards Organization*. ISO/IEC IS 8879. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). 1986.

(JAVA, 2000a) *The Java Language: An Overview*. Abril 2000.
<http://java.sun.com/docs/overviews/java/java-overview-1.html>

(JAVA, 2000b) *The Source for JavaTM Technology*. Abril 2000. <http://java.sun.com>

(JAVASPACEs, 2000a) *JavaSpacesTM Service Specification*. Outubro 2000.

<http://www.sun.com/jini/specs/jini1.1.html/js-title.html>

(JAVASPACEs, 2000b) *JavaSpacesTM Technology*. Outubro 2000.

<http://java.sun.com/products/javaspaces/>

(JINI, 2000) *JiniTM Technology Core Platform Specification*. Outubro 2000.

<http://www.sun.com/jini/specs/jini1.1.html/core-title.html>

(JINI, 2001) *Jini Network Technology*. Fevereiro 2001. <http://www.sun.com/jini/>

(Johnson, 1999) Johnson, M. *XML for the Absolute Beginner*. Abril 1999.

http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml_p.html

(Korth & Silberschatz, 1993) Korth, H. F.; Silberschatz, A.; *Sistema de Banco de Dados*. Tradução: Maurício Heihachiro Galvan Abe. Revisão Técnica: Sílvia Carmo Palmieri. Segunda Edição. Makron Books, 1993.

(Le Hors et al., 2000) Le Hors, A.; Le Hegaret, P.; Wood, L.; Nicol, G.; Robie, J.; Champion, M.; Byrne, S.; *Document Object Model (DOM) Level 2 Core Specification Version 1.0*. Novembro 2000. <http://www.w3.org/TR/DOM-Level-2-Core/>

(Light, 1999) Light, R. *Iniciando em XML*. Makron Books, 1999.

(Mace et al., 1998) Mace, S.; Flohr, V.; Dobson, R.; Graham, T. *Weaving a Better Web*. Byte, v. 23, n. 3, pp. 58-68, 1998.

(McGrath, 1999) McGrath, S. *XML Aplicações Práticas - Como Desenvolver Aplicações de Comércio Eletrônico*. Editora Campus, 1999.

(Megginson, 1998) Megginson, D. *SAX 1.0: The Simple API for XML*. Maio 1998.
<http://www.megginson.com/SAX/SAX1/index.html>

(Megginson, 2000) Megginson, D. *SAX 2.0: The Simple API for XML*. Maio 2000.
<http://www.megginson.com/SAX/index.html>

(Melo et al., 1997) Melo, R. N.; Silva S. D.; Tanaka, A. K. *Banco de Dados em Aplicações Cliente-Servidor*. Infobook, 1997.

(Naughton, 1996) Naughton, P. *Dominando o JAVA*. Makron Books, 1996.

(Raggett et al., 1999) Raggett, D.; Le Hors, A.; Jacobs, I. *HTML 4.01 Specification*.
Dezembro 1999. <http://www.w3.org/TR/REC-html40/>

(Robie, 1999a) Robie, J. *XQL Tutorial*. Março 1999. <http://www.ibiblio.org/xql/xql-tutorial.html>

(Robie, 1999b) Robie, J. *XQL (XML Query Language)*. Agosto 1999.
<http://www.ibiblio.org/xql/xql-proposal.html>

(Robie et al., 1998) Robie, J.; Lapp, J.; Schach, D.; Hyman, M.; Marsh, J. *XML Query Language (XQL)*. Setembro 1998. <http://www.w3.org/TandS/QL/QL98/pp/xql.html>

(Van Hoff, 1998) Van Hoff, A. *Java: Getting Down to Business*. Dr. Dobb's Journal.
Janeiro 1998, 20-24.

(Wood & Le Hégarret, 2000) Wood, L.; Le Hégarret, P. *Document Object Model (DOM)*.

Março 2000. <http://www.w3.org/DOM/>

(XALAN, 2000) *Xalan-Java Overview*. Novembro 2000. <http://xml.apache.org/xalan-j/overview.html>

(XERCES, 2000) *Xerces Java Apache Readme*. Novembro 2000.

<http://xml.apache.org/xerces-j/index.html>