

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

IVONEI FREITAS DA SILVA

**CONCEITOS DE REDES ATIVAS EM SISTEMAS
DE OBJETOS DISTRIBUÍDOS**

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


Prof. Dr. João Bosco Manguiera Sobral

Florianópolis, dezembro de 2000

CONCEITOS DE REDES ATIVAS EM SISTEMAS DE OBJETOS DISTRIBUÍDOS

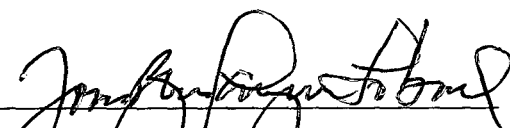
Ivonei Freitas da Silva

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

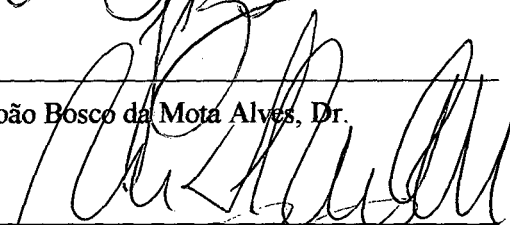


Prof. Fernando A. O. Gauthier

Banca Examinadora



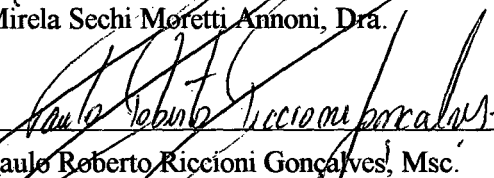
João Bosco Mangueira Sobral, Dr., Orientador



João Bosco da Mota Alves, Dr.



Mirela Sechi Moretti Annoni, Dra.



Paulo Roberto Riccioni Gonçalves, Msc.

“Não se pode ensinar alguma coisa a
alguém, pode-se apenas auxiliá-lo a
descobrir por si mesmo”.

Galileu Galilei

AGRADECIMENTOS

A UFSC-CTC-CPGCC-INE, pela oportunidade.

Aos professores, pelo ensinamento.

A banca, pela contribuição.

Ao meu irmão Claudinei, aos amigos Euclides de Barros Jr., Luciano Nakano, Cidão e Valmir Gonçalves, pelo apoio.

Aos meus filhos, pela inspiração.

A Verinha e a Valdete, pela atenção.

Ao meu orientador, João Bosco M. Sobral, por tudo.

SUMÁRIO

LISTA DE FIGURAS	IX
LISTA DE QUADROS	XII
LISTA DE TABELAS.....	XIII
LISTA DE ABREVIATURAS E SIGLAS	XIV
RESUMO.....	XVI
ABSTRACT	XVII
1. INTRODUÇÃO.....	1
1.1. OBJETIVO.....	2
1.1.1. <i>Objetivos gerais</i>	2
1.1.2. <i>Objetivos específicos</i>	3
1.2. MOTIVAÇÃO	3
1.3. TRABALHOS CORRELATOS	5
1.4. ESTADO DA ARTE	5
1.5. ESTRUTURA DA DISSERTAÇÃO	6
2. CONCEITOS DE REDES ATIVAS.....	8
2.1. CARACTERIZAÇÃO DE REDES ATIVAS	8
2.2. TÉCNICA DISCRETA (PROGRAMMABLE NODE)	10
2.3. TÉCNICA INTEGRADA (CAPSULES)	10
2.4. COMO FICA A INFRAESTRUTURA EXISTENTE?	10
2.5. ARQUITETURA DO NODO ATIVO	11
2.6. COMPONENTES FUNDAMENTAIS	12
2.7. ARMAZENAGEM ATIVA	12
2.8. EXTENSIBILIDADE	12

2.9. BENEFÍCIOS DAS REDES ATIVAS	15
2.9.1. <i>Disponibilidade de informações seguras pelos nodos intermediários</i>	16
2.9.2. <i>Capacidade de processamento de dados durante o caminho</i>	16
2.9.3. <i>Adoção de estratégias distribuídas</i>	16
2.9.4. <i>Facilidade no desenvolvimento de novos serviços de rede</i>	16
2.10. CONSIDERAÇÕES FINAIS	17
3. CONCEITOS DE REDES ATIVAS NO NÍVEL DE APLICAÇÃO	18
3.1. SERVIDOR DE <i>PROXY</i> DINÂMICO (SPD)	18
3.2. OS PROGRAMAS ESPECÍFICOS (PROXYLETS)	19
3.3. O SERVIDOR DE PROTOCOLO DINÂMICO	19
3.4. ASPECTOS DE IMPLEMENTAÇÃO	20
3.4.1. <i>Servidor de proxy dinâmico</i>	20
3.4.2. <i>Proxylet</i>	21
3.5. A ARQUITETURA DE CONTROLE DO RANA	21
3.6. A <i>INTERFACE</i> DE CONTROLE	22
3.7. A <i>INTERFACE</i> DE MONITOR	22
3.8. ASPECTOS DE SEGURANÇA	22
4. XML COMO UMA TECNOLOGIA PARA ABSTRAÇÃO DOS CONCEITOS DE REDES ATIVAS NO NÍVEL DE APLICAÇÃO	24
4.1. INTRODUÇÃO A XML	24
4.2. CARACTERÍSTICAS DE XML	25
4.3 VANTAGENS DE SE DEFINIR AS PRÓPRIAS <i>TAGS</i>	26
4.4. DOCUMENTO DE TIPO DE DADOS (DTD)	28
4.5. FORMATAÇÃO	29
4.6. XML NA PRÁTICA	30
4.6.1. <i>Documentos válidos e bem-formatados</i>	30
4.6.2. <i>Regras de formação XML</i>	32
4.6.3. <i>Escrevendo uma aplicação</i>	35
4.7. VANTAGENS DA XML	44
4.7.1. <i>Informação no cabeçalho</i>	45
4.7.2. <i>Classes de documentos</i>	45

4.7.3. Documentos longos.....	46
4.7.4. Foco na informação.....	46
4.7.5. Estrutura que pode ser pesquisada	47
4.8. APLICAÇÕES E FERRAMENTAS	48
4.8.1. Jumbo.....	48
4.8.2. MathML.....	49
4.8.3 Ferramentas de edição	50
4.9. CONSIDERAÇÕES FINAIS	52
5. CORBA COMO UMA TECNOLOGIA PARA ABSTRAÇÃO DOS CONCEITOS DE REDES ATIVAS NO NÍVEL DA APLICAÇÃO	54
5.1. INTRODUÇÃO AO PADRÃO CORBA	54
5.2. UMA VISÃO ARQUITETURAL DO PADRÃO CORBA	55
5.2.1. <i>Object management group (OMG)</i>	55
5.2.2. <i>A aplicação cliente</i>	55
5.2.3. <i>Invocação estática (STUB)</i>	56
5.2.4. <i>Invocação dinâmica</i>	56
5.2.5. <i>Object request broker (ORB)</i>	57
5.2.6. <i>Contexto de objeto</i>	58
5.2.7. <i>Repositório de interface</i>	59
5.2.8. <i>Interface definition language (IDL)</i>	60
5.2.9. <i>Aplicação servidora</i>	63
5.2.10. <i>Adaptador de objeto</i>	64
5.2.11. <i>Skeleton do servidor</i>	65
5.2.12. <i>Repositório de Implementação</i>	66
5.3. INTEROPERABILIDADE EM CORBA	66
5.4. SERVIÇOS E FACILIDADES DO PADRÃO CORBA	67
5.5. CONSIDERAÇÕES FINAIS DO PADRÃO CORBA	68
6. DESENVOLVENDO O FRAMEWORK E A APLICAÇÃO	69
6.1 ESPECIFICAÇÃO DO FRAMEWORK EM IDL.....	70
6.2 CONSIDERAÇÕES SOBRE A ESPECIFICAÇÃO	71
6.3 A APLICAÇÃO.....	71

6.3.1 <i>Justificativa</i>	72
6.4. MATERIAIS E MÉTODO PARA CONSTRUÇÃO DA APLICAÇÃO	75
6.4.1. <i>Ferramentas utilizadas</i>	76
6.4.2. <i>Ambiente de desenvolvimento</i>	76
6.4.3. <i>A aplicação na prática</i>	77
6.4.4. <i>Teste da funcionalidade da aplicação em execução</i>	78
6.4.5. <i>Resultados: Medindo o desempenho</i>	83
6.4.6. <i>Análise dos resultados</i>	85
7. CONCLUSÃO FINAL	87
7.1. CONSIDERAÇÕES INICIAIS	87
7.2. RESULTADOS ALCANÇADOS	88
7.3. PERSPECTIVAS	88
7.4. TRABALHOS FUTUROS	89
8. REFERÊNCIAS BIBLIOGRÁFICAS	90
ANEXO 1: MODELAGEM DA APLICAÇÃO DE CONSULTA DE DADOS XML NA LINGUAGEM UML	94
ANEXO 2: CODIFICAÇÃO NA LINGUAGEM JAVA DA APLICAÇÃO DE CONSULTA DE DADOS XML	99
ANEXO 3: QUADRO QUE LISTA O CONTEÚDO DO ARQUIVO XML UTILIZADO PELA APLICAÇÃO	111
ANEXO 4: TABELAS DE MENSURAÇÃO DOS TEMPOS DE RESPOSTA DAS APLICAÇÕES	114

LISTA DE FIGURAS

FIGURA 1:DISPOSITIVOS LEGADOS/ATIVOS EM UMA REDE ATIVA.....	9
FIGURA 2: ORGANIZAÇÃO DE UM NODO ATIVO.....	12
FIGURA 3: EXEMPLO DE UMA TOPOLOGIA PARA REDE ATIVA VIRTUAL	13
FIGURA 4: MODELO DE NODO ATIVO PROPOSTO POR A. B. KULKARNI	14
FIGURA 5: FORMATO DO <i>SMARTPACKET</i>	15
FIGURA 6: VISÃO DA ARQUITETURA RANA.....	20
FIGURA 7: RELAÇÃO ENTRE SGML, PCIS, EDGAR E HTML.....	25
FIGURA 8: DOCUMENTO HTML SENDO EXIBIDO NA WEB	27
FIGURA 9: DOCUMENTO XML SENDO EXIBIDO NA WEB	27
FIGURA 10: A RELAÇÃO ENTRE PADRÕES DE ESTILO E DOCUMENTAÇÃO	30
FIGURA 11: CONTEÚDO, ESTRUTURA E APRESENTAÇÃO INTEGRADOS PARA EXIBIÇÃO NA <i>WEB</i>	30
FIGURA 12: ESTRUTURA LÓGICA DE UM ARQUIVO XML.....	31
FIGURA 13: ESTRUTURA FÍSICA DE UM DOCUMENTO XML.....	32
FIGURA 14: ARQUIVO PLANTAS.XML ABERTO ATRAVÉS DO MICROSOFT INTERNET EXPLORER 5.0.....	38

FIGURA 15: ARQUIVO PLANTAS.XML ABERTO ATRAVÉS DO INTERNET EXPLORER 5.0	41
FIGURA 16: ARQUIVO PLANTAS.XML ABERTO ATRAVÉS DO MICROSOFT INTERNET EXPLORER 5.0.....	43
FIGURA 17: UMA MOLÉCULA REPRESENTADA NA APLET JUMBO. À ESQUERDA TEM-SE O CÓDIGO FONTE DO ARQUIVO.....	49
FIGURA 18: TELA DO EXML	50
FIGURA 19: TELA DO XMLWRITER.....	51
FIGURA 20: TELA DO IBM XSL EDITOR.....	52
FIGURA 21: ARQUITETURA DE UM SISTEMA CORBA.....	55
FIGURA 22: ORB SELECIONANDO UMA IMPLEMENTAÇÃO	58
FIGURA 23: ARQUIVOS QUE PODEM SER GERADOS PELO COMPILADOR IDL.....	62
FIGURA 24: ESTRUTURA BÁSICA DE UMA APLICAÇÃO SERVIDORA	64
FIGURA 25: RELAÇÕES DOS PROTOCOLOS INTER-ORB	67
FIGURA 26: VISÃO DOS COMPONENTES DA APLICAÇÃO	74
FIGURA 27: PASSOS QUE DESCREVEM COMO UMA CONSULTA GERA UM RESULTADO AO USUÁRIO.....	75
FIGURA 28: ARQUIVO BOOK-ORDER.XML VISUALIZADO ATRAVÉS DO MICROSOFT INTERNET EXPLORER 5.0.....	79
FIGURA 29: ÍCONE DO SMART AGENT	79
FIGURA 30: COMANDO QUE IRÁ INICIALIZAR O SERVIDOR DE OBJETOS.....	80

FIGURA 31: TELA DO SERVIDOR.....	80
FIGURA 32: CLIENTE SENDO EXECUTADO	81
FIGURA 33: ARQUIVO QUE RESULTOU DA CONSULTA EFETUADA	81
FIGURA 34: EXEMPLO DE UMA CONSULTA UTILIZANDO CHARACTER CURINGA.....	82
FIGURA 35: ARQUIVO RQ2.XML GERADO ATRAVÉS DE UMA CONSULTA AO SERVIDOR	82
FIGURA 36: DIAGRAMA DE USE-CASE DA APLICAÇÃO	94
FIGURA 37: DIAGRAMA DAS CLASSES DA APLICAÇÃO.....	95
FIGURA 38: DIAGRAMA DE SEQUÊNCIA DA APLICAÇÃO	96
FIGURA 39: DIAGRAMA DOS COMPONENTES FÍSICOS DA APLICAÇÃO	97
FIGURA 40: DIAGRAMA DE EXECUÇÃO DA APLICAÇÃO	98

LISTA DE QUADROS

QUADRO 1: TRECHOS DE DOCUMENTOS HTML E XML.....	26
QUADRO 2: MODELO DA ESTRUTURA DO ARQUIVO XML	35
QUADRO 3: DTD DO ARQUIVO DE PLANTAS.....	36
QUADRO 4: ARQUIVO PLANTAS.XML.....	37
QUADRO 5: RESULTADO DO TESTE DO ARQUIVO PLANTAS.XML ATRAVÉS DA APLICAÇÃO DE SUA DTD	38
QUADRO 6: ARQUIVO DE ESTILO PLANTAS.XSL.....	39
QUADRO 7: NOVO ARQUIVO PLANTAS.XML	40
QUADRO 8: NOVO ARQUIVO PLANTAS.XSL GERADO COM O PROPÓSITO DE IMPOR UMA NOVA ORDENAÇÃO AO ARQUIVO PLANTAS.XML.....	42
QUADRO 9: LISTAGEM DE CÓDIGO NA LINGUAGEM IDL	63
QUADRO 10: LISTAGEM DA ESPECIFICAÇÃO DO FRAMEWORK.....	70
QUADRO 11: LISTAGEM DO CÓDIGO DA APLICAÇÃO CLIENTE.....	101
QUADRO 12: LISTAGEM DO CÓDIGO DA APLICAÇÃO SERVIDORA	103
QUADRO 13: LISTAGEM DO CÓDIGO DA CLASSE QUE IMPLEMENTA O SERVIÇO PESQUISA/FILTRAGEM DE DADOS XML	110
QUADRO 14: LISTAGEM DO ARQUIVO BOOK-ORDER.XML.....	113

LISTA DE TABELAS

TABELA 1: COMPARAÇÃO ENTRE CÓDIGO HTML E CÓDIGO XML QUE MOSTRA COMO XML É MAIS FORMAL.....	32
TABELA 2: COMPARAÇÃO ENTRE CÓDIGO HTML E XML. ANINHAMENTO DOS ELEMENTOS XML EM FORMA DE ÁRVORE.....	33
TABELA 3: COMPARAÇÃO ENTRE CÓDIGO HTML E XML. É SUGERIDO QUE AS TAG'S XML SÃO SEMPRE MAIÚSCULAS OU MINÚSCULAS	33
TABELA 4: COMPARAÇÃO ENTRE A LINGUAGEM HTML E A LINGUAGEM XML. ATRIBUTOS EM XML ENTRE ASPAS.....	33
TABELA 5: COMPARAÇÃO ENTRE HTML E XML. HÁ APENAS UM ELEMENTO PRINCIPAL EM XML.....	34
TABELA 6: COMPARAÇÃO ENTRE LINGUAGEM HTML E XML. EM XML SCRIPTS SÃO ESCRITOS EM BLOCO CDATA PARA NÃO ATRAPALHAR O PROCESSAMENTO DOS COMANDOS XML.....	34
TABELA 7: TABELAMENTO DOS TEMPOS COM SUAS RESPECTIVAS CONFIGURAÇÕES	84
TABELA 8: TEMPOS DE RESPOSTA COM ARQUIVOS GRANDES.....	114
TABELA 9: TEMPOS DE RESPOSTA COM ARQUIVOS PEQUENOS	114

LISTA DE ABREVIATURAS E SIGLAS

API *Application Program Interface*

BOA *Basic Object Adapter*

CML *Chemical Markup Language*

CORBA *Common Object Request Broker Architecture*

CSS *Cascading Style Sheets*

DARPA *Defense Advanced Research Project Agency*

DSSSL *Document Style Semantics and Specification Language*

DTD *Documento de Tipo de Dados*

GIOP *General Inter-ORB Protocol*

HTML *HyperText Markup Language*

IDL *Interface Definition Language*

HOP *Internet Inter-ORB Protocol*

IP *Internet Protocol*

ISO *International Organization for Standardization*

JAR *Java Archive*

JDK *Java Development Kit*

JVM *Java Virtual Machine*

MOM *Message-Oriented Middleware*

OMG *Object Management Group*

ORB *Object Request Broker*

OSI *Open System Interface*

RA *Redes Ativas*

RANA *Redes Ativas ao Nível de Aplicação*

RFC *Request For Comments*

RMI *Remote Method Invocation*

RPC *Remote Procedure Call*

SGML *Standard Generalized Markup Language*

SPD *Servidor de proxy Dinâmico*

TCP *Transport Control Protocol*

URL *Uniform Resource Locator*

WWW *World Wide Web*

XML *eXtensible Markup Language*

XSL *XML Style Language*

RESUMO

A atual infraestrutura de redes de transmissão de dados não suporta adequadamente os novos serviços para aplicações na *Internet*.

Diante deste fato, este trabalho busca um estudo de novas tecnologias como alternativa para o desenvolvimento de novos serviços na infraestrutura de redes existente.

Rede Ativa ao Nível da camada da Aplicação do protocolo OSI da ISO, XML e CORBA mostram-se muito valiosas para o desenvolvimento de novos serviços para os sistemas distribuídos.

Este trabalho visa um estudo teórico e prático destas tecnologias, com o intuito de desenvolver aplicações que diminuam o tráfego de dados entre uma máquina cliente e uma máquina servidora. Para isso, foi implementada uma aplicação que diminui o tráfego de dados utilizando estas tecnologias dentro do contexto das Redes Ativas ao Nível de Aplicação.

ABSTRACT

The current infrastructure of networks of data transmission doesn't support the new services appropriately for Internet applications.

Before this fact, this work search a study of new technologies as alternative for the development of new services in the existent infrastructure of nets.

Application Level Active Networking of the protocol OSI of ISO, XML and CORBA they are shown very valuable for the development of new services for the distributed systems.

This work seeks a theoretical and practical study of these technologies, with the objective of developing applications that decrease the traffic of data between a machine customer and a machine server. For that, it was implemented an application that decreases the traffic of data using these technologies inside of the context of the Application Level Active Networking.

1. INTRODUÇÃO

Este trabalho abstrai conceitos de Redes Ativas na camada da aplicação do protocolo OSI da ISSO, utilizando as tecnologias CORBA, XML e Java para o desenvolvimento de aplicações de objetos distribuídos na *WEB*.

As aplicações de tráfego de voz, imagens e gerenciamento de redes são exemplos que exigem maior flexibilidade da infraestrutura de comunicação de dados. As redes de transmissão de dados atuais não conseguem atender adequadamente os novos requisitos dessas aplicações. Faz-se necessário o desenvolvimento de novos modelos e aplicações que supram naturalmente esses requisitos e que sejam base para o desenvolvimento de novos serviços. O paradigma de Redes Ativas (RA) promete ser um caminho flexível e dinâmico na introdução dessas novas funcionalidades na rede.

Através do aumento da computação nos nodos da rede, poder-se-á atualizar e personalizar o processamento na rede para diversos requisitos das aplicações.

No entanto, Redes Ativas não é um conceito totalmente novo. Os serviços de filtros de pacotes em *firewalls*, de *WEB proxies* e roteadores *multicast* são exemplos de aplicações que agregam computação em nodos intermediários entre uma comunicação ponto-a-ponto.

Observa-se que a Rede Ativa proposta inicialmente em [TEN 96] está em um processo embrionário. Por este motivo, muitas questões ainda não foram respondidas.

Diante do exposto, surge uma ramificação desse paradigma que se apresenta mais realizável ao curto prazo. Essa subdivisão denomina-se de **Redes Ativas ao Nível de Aplicação (RANA)**. Essas redes explicitam o paradigma de Redes Ativas na camada da aplicação do protocolo OSI em forma de aplicações. Elas consistem de um ambiente com clientes e servidores, tal como, os *browsers* e servidores *WEB* em uma *Internet*. A principal diferença é a inserção de servidores de *proxy* dinâmicos (SPD) entre a

comunicação de clientes e servidores. O SPD quando colocado em pontos estratégicos da rede pode melhorar a comunicação cliente/servidor fornecendo novos serviços.

Mas, para que os conceitos de redes ativas sejam realizáveis necessita-se de tecnologias que implementem tais conceitos de forma distribuída e estruturada, pois o ambiente computacional em questão constitui-se de diversas arquiteturas de computadores, sistemas operacionais e um variado conjunto de informações, tudo isto interconectado em uma grande rede computacional.

A linguagem XML (eXtensible Markup Language) e o padrão CORBA (Common Object Request Broker Architecture) são essenciais para abstrair os conceitos de Redes Ativas nas aplicações da camada superior do protocolo OSI da ISO. A linguagem XML possui como uma de suas principais características a estruturação da informação que carrega consigo, justamente o necessário para a otimização de processamentos em seu conteúdo, como por exemplo, a filtragem e a pesquisa de dados. O padrão CORBA dita as regras necessárias para a comunicação transparente entre diversas plataformas computacionais, além de impor um ambiente computacional distribuído, sendo este um dos conceitos de uma rede ativa.

1.1. Objetivo

Esse trabalho aborda um estudo amplo, mas não exaustivo, do paradigma de Redes Ativas bem como um estudo detalhado de Redes Ativas ao Nível de Aplicação, proposta inicialmente em [FRY 98]. Também está presente um estudo das tecnologias XML e CORBA, as quais possuem características essenciais para o desenvolvimento de aplicações distribuídas, interoperáveis e com informações estruturadas em uma rede de computadores.

1.1.1. Objetivos gerais

Analisar as características relevantes das tecnologias Redes Ativas, XML e CORBA;

Relatar as vantagens destas tecnologias no desenvolvimento de aplicações para a *WEB*;

Abstrair os conceitos de redes ativas em aplicações para *WEB*, através da linguagem XML e do padrão CORBA;

1.1.2. Objetivos específicos

Desenvolver uma aplicação que pesquise dados na *WEB* através das tecnologias mencionadas;

Diminuir o tráfego de dados da rede com a aplicação de pesquisa de dados no ambiente XML/CORBA/JAVA comparado com outra tecnologia HTML/JAVA;

Mostrar com a aplicação desenvolvida as vantagens de tais tecnologias;

Documentar esta aplicação através de um modelo em UML (Unified Modeling Language);

Desenvolver um *Framework* para busca de dados ao nível de especificação na linguagem IDL/CORBA;

1.2. Motivação

A Internet é um meio eficiente para divulgação de informações, onde cada vez mais empresas e pessoas têm se utilizado dela para divulgar e coletar dados. Por este motivo, seu crescimento tem sido vertiginoso e isto gerou um problema: pesquisar informações específicas na rede é, atualmente, uma tarefa complexa.

O modelo de pesquisa utilizado na Internet é bastante simples: efetua-se uma consulta textual a uma base de dados. Contudo, este modelo de pesquisa possui várias restrições, dentre eles: a base de dados pode não dispor de todos os *sites* da Internet e a informação de que precisamos não está disponível naquela base de dados. Um outro problema é o volume de informações que retornam, onde a maioria destas informações não se encaixa no que precisamos. Isto decorre do fato da pesquisa ser puramente textual.

As *Intranets* criadas em grandes empresas possuem os mesmos problemas. O volume de informações colocadas à disposição dos usuários é muito grande e a forma de busca dessas informações já não é satisfatória.

Outro aspecto importante a ser relatado está baseado no problema do gerenciamento das informações dentro de uma empresa. O custo de produzir, agrupar e manter as informações necessárias sobre um produto ou uma decisão pode exceder o custo inicial das máquinas e dos programas.

Para as empresas, a informação é parte dos produtos e das decisões relacionados ao cliente e, portanto, precisa ser rigorosamente mantida. Geralmente, essas informações existem, mas estão desestruturadas e dispersas em várias formas de armazenamento.

A linguagem XML ameniza estes problemas. Sua estrutura é perfeita para descrever a informação que o documento contém. Por conta disto, pode-se fazer uma pesquisa específica muito mais facilmente e encontrar resultados muito mais satisfatórios, praticamente eliminando aqueles que não nos interessam. A linguagem XML também pode nos retornar apenas a informação que se procura, (um dos conceitos da rede ativa), que por sua vez, implica em um menor tráfego na rede (um dos objetivos da rede ativa).

Os conceitos de redes ativas na camada de rede do modelo de referência OSI da ISO podem ser abstraídos em aplicações sobre a camada da aplicação. Na verdade, há inúmeras aplicações que absorvem esses conceitos os quais podem contribuir para o aumento de desempenho da rede.

Portanto, a criação de uma rede ativa de informações objetiva gerenciar adequadamente esses dados que são de vital importância para a empresa, baseando-se em tecnologias e conceitos que possuem características como a estruturação, a interoperabilidade e a distribuição dos dados em diversos ambientes computacionais.

A Internet tornou-se motivação e berço de várias novas tecnologias (Redes Ativas, XML, JAVA, CORBA), assim é necessário estudá-las para, através destas tecnologias, ordenar e otimizar a enorme quantidade de informações disponível na rede. Deste modo, a integração destas tecnologias pode prover serviços que otimizem os recursos computacionais de uma rede de computadores.

1.3. Trabalhos correlatos

Alguns trabalhos alicerçam a concepção desta dissertação. Esses estão referenciados no texto e relacionados no capítulo Referências Bibliográficas.

Em [TEN 96] é mostrado a base e os conceitos iniciais sobre redes ativas.

Em [FRY 98] é apresentado um trabalho amplo sobre redes ativas na camada da aplicação, realizado na Universidade de Tecnologia de Sydney, Austrália. Esse trabalho foi o grande incentivador desta dissertação.

Em [MAR 99] está um trabalho apresentado na conferência internacional sobre redes ativas (IWAN'99) que ratificou a importância da abstração dos conceitos de redes ativas na camada da aplicação do protocolo OSI da ISO.

Um outro trabalho está em [KUL 97], no qual o autor sugere uma implementação para redes ativas na plataforma Java, desenvolvido na Universidade do Kansas, Estados Unidos.

1.4. Estado da Arte

O desenvolvimento de aplicações distribuídas na Internet está a cada dia, utilizando-se de tecnologias que melhoram o nível de serviço existente, bem como, agregam mais serviços, dantes impensados pela falta de tais tecnologias.

A linguagem XML está extrapolando seus próprios limites integrando-se a outras tecnologias como a própria linguagem HTML e a linguagem de modelagem orientada a objetos UML.

A W3C é um consórcio que também centraliza diversos trabalhos sobre a linguagem XML. O projeto XHTML é uma iniciativa que agrega os aspectos positivos das linguagens XML e HTML, sendo uma recomendação da W3C em sua versão básica. Esse projeto visa a aplicação em aparelhos eletrodomésticos servindo de ponte para a comunicação com o computador [W3C00].

Diversos trabalhos em Redes Ativas estão sendo realizados em inúmeros centros de pesquisa. Nota-se os trabalhos da Universidade da Pensilvânia com o projeto

SwitchWare, do qual surge a criação da linguagem de *script* PLAN (Packet Language for Active Network), cujos programas são transportados em pacotes e executados em máquinas na rede. Na criação de linguagens, protocolos e plataformas existem outros centros de pesquisas que estão trabalhando para prover ao médio prazo modelos e ferramentas para as redes ativas ao nível da camada de rede do protocolo OSI da ISO.

O principal trabalho em redes ativas ao nível de aplicação encontra-se na Universidade de Tecnologia em Sydney, Austrália. Este trabalho visa a construção de uma arquitetura de *software* que fornece as mesmas características das redes ativas ao nível de rede, além de contornar os problemas existentes nestas redes ativas.

Com relação ao padrão CORBA, [ORF 98] descreve várias aplicações que podem ser desenvolvidas com esta arquitetura. Algumas especificações do padrão CORBA são adotadas por organizações na área da computação, como exemplo tem-se a ISO/IEC 14750 adotando a linguagem IDL. Há vários consórcios e organizações que utilizam a arquitetura CORBA, além da linguagem UML, para o desenvolvimento de padrões [COR 00].

1.5. Estrutura da dissertação

Este documento está subdividido em capítulos que abordam os seguintes conteúdos:

- Capítulo 2: descreve os conceitos, a arquitetura e os benefícios que uma Rede Ativa pode fornecer;
- Capítulo 3: descreve os conceitos, a arquitetura e as vantagens que uma Rede Ativa no Nível de Aplicação pode fornecer;
- Capítulo 4: descreve a tecnologia XML para abstração dos conceitos das redes ativas;
- Capítulo 5: descreve a tecnologia CORBA para abstração dos conceitos das redes ativas;
- Capítulo 6: descreve as características da aplicação desenvolvida, os materiais e os métodos utilizados para o desenvolvimento e o *framework*

proposto que serve de base para o desenvolvimento de futuras aplicações. Há também considerações sobre o resultados adquiridos com a aplicação em execução considerando as tecnologias adotadas para abstrair conceitos de redes ativas na camada de aplicação do protocolo OSI;

- Capítulo 7: descreve as conclusões inferidas do trabalho, além dos resultados alcançados. Também estão descritas sugestões para futuros trabalhos na mesma linha de pesquisa;
- Capítulo 8: refere-se às referências bibliográficas que este trabalho está fundamentado;
- Anexo 1: apresenta a modelagem na linguagem UML da aplicação.
- Anexo 2: apresenta os principais aspectos da codificação da aplicação.
- Anexo 3: apresenta a listagem de um arquivo XML utilizado para testar as funcionalidades e o desempenho da aplicação.
- Anexo 4: apresenta os tempos de resposta das aplicações em forma tabular.

2. CONCEITOS DE REDES ATIVAS

Este capítulo aborda a conceituação sobre a tecnologia de Redes Ativas, descrevendo sua arquitetura e os benefícios que essa pode oferecer para futuras aplicações.

2.1. Caracterização de Redes Ativas

A atual arquitetura das redes de transmissão de dados não é adequada para as novas necessidades e os novos serviços de rede identificados.

Uma característica que se observa neste atual modelo de arquitetura de rede é a computação limitada nos diversos nodos¹ entre o ponto inicial e o ponto final de uma transmissão de dados. Essas redes transportam *bits* passivamente nos nodos intermediários de uma comunicação ponto-a-ponto. O limitado processamento realizado nesses nodos refere-se apenas a computação do cabeçalho nos pacotes e a sinalização em redes orientadas à conexão.

Entre 1994 e 1995 o departamento *Defense Advanced Research Project Agency* de sistemas de informação automatizado (DARPA) [DAR 00], relatou diversos problemas com a atual arquitetura de rede, dentre eles: as dificuldades de integração de novos padrões e tecnologias em uma infraestrutura de rede compartilhada, as restrições de *performance* devido à redundância de operações nas várias camadas do protocolo de comunicação e a dificuldade de inserir novos serviços no modelo arquitetural existente.

Baseado nessas discussões a comunidade DARPA identificou estratégias que convergiram no sentido de aumentar a computação nos diversos nodos da rede.

¹ Nodos: dispositivos, tais como: roteadores e *switches*

Com os primeiros estudos sobre o aumento na computação nos nodos da rede, detectou-se novos serviços que eram inimagináveis com as redes de dados tradicionais. Como exemplo, tem-se a sessão *multicast* de vídeo onde, em cada um dos nodos, é modificado o esquema de compressão de vídeo baseado na computação feita por aquele nodo e na largura de banda disponível. Logo, os nodos de uma **rede programável** não apenas transportam *bits*, mas também **processam** ou **gerenciam** esses dados. Os dispositivos que efetuam este tipo de processamento são denominados de **nodo ativo**, e, ao considerarmos uma rede composta de nodos ativos, teremos uma **Rede Ativa**.

O conceito de Redes Ativas emergiu com a inclusão de computação nos diversos nodos da rede. Em [TEN 96] as redes ativas são visualizadas como um conjunto de nodos ativos que efetuam operações personalizadas sobre os dados transportados nessas redes.

Redes ativas não somente permitem aos nodos da rede efetuar computações sobre os dados, mas também permitem injetar programas personalizados dentro desses pontos da rede, os quais podem modificar, armazenar ou redirecionar o fluxo de dados na rede.

A Fig. 1 ilustra uma rede com nodos ativos e nodos legados.

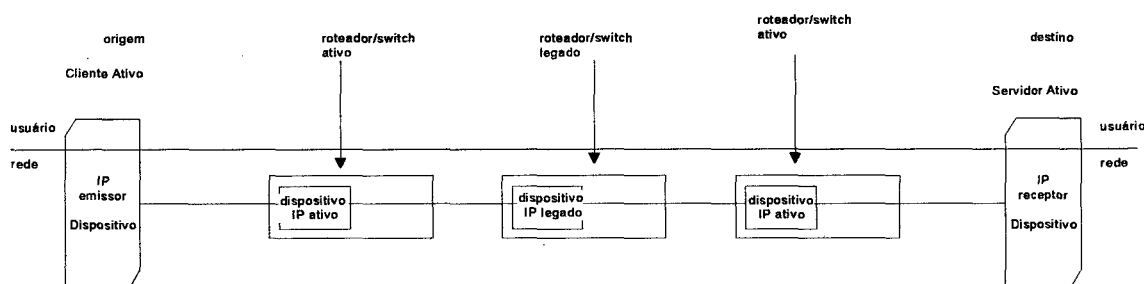


FIGURA 1:DISPOSITIVOS LEGADOS/ATIVOS EM UMA REDE ATIVA

Contudo, para que essa arquitetura seja efetivada é preciso fornecer técnicas que facilitem a injeção e a execução de programas nos nodos intermediários da rede.

Para a injeção de programas, duas técnicas até o momento são constantemente relatadas nos trabalhos de pesquisa na área. As técnicas discreta e integrada, explicadas nas próximas seções.

2.2. Técnica discreta (programmable node)

Existem dois tipos de pacotes que trafegam através dos nodos em uma rede. Os pacotes que contém dados e os pacotes que contém programas.

Diante disto, surge a técnica discreta, na qual assume-se que os pacotes que transportam programas irão atuar sobre os pacotes que transportam dados.

Nesta técnica, os programas são injetados dentro dos nodos ativos programáveis separadamente dos pacotes de dados que caminham dentro da rede. Desta forma, primeiro injeta-se as instruções dentro dos nodos; então, envia-se os dados que podem ser submetidos ou não ao(s) programa(s) que estão nos nodos ativos.

O projeto *SwitchWare* [SCO 98] é um exemplo que trata desta técnica.

2.3. Técnica integrada (capsules)

Como sugere o próprio nome (técnica integrada), os dados e os programas estão integrados no mesmo pacote, denominado no contexto de redes ativas de **cápsulas**.

As cápsulas possuem um fragmento de programa que pode incluir dados embutidos. Quando uma cápsula chega em um nodo, seu conteúdo é avaliado, da mesma forma que ocorre com as impressoras *PostScript* que interpretam o conteúdo dos arquivos que chegam para serem impressos.

2.4. Como fica a infraestrutura existente?

Uma questão importante a ser levantada é a interpretação dos nodos da rede ativa juntamente com os nodos da 'rede legada'. Felizmente, o nodo ativo pode co-existir na rede com o nodo legado através da técnica de tunelamento².

² Tunelamento: ligações virtuais ponto a ponto que podem ser implementadas por uma pilha de protocolo separada.

Há também uma proposta em RFC emitida em [ALEX RFC] que esboça um possível mecanismo para encapsulação de *frames* de rede ativa. O protocolo proposto permite o uso da infraestrutura de rede existente numa rede ativa.

Outro trabalho proposto em [WET 96] propõe uma extensão ao protocolo IP que reforça as capacidades ativas para a *Internet* existente. O campo *options* do protocolo IP já está sendo usado para incorporar novas capacidades para a rede IP, principalmente para aplicações que envolvem a mensuração e o monitoramento de redes.

De qualquer forma, para que as técnicas integrada e discreta atinjam seus objetivos, é necessário que elas suportem três aspectos essenciais [TEN 97]:

- **Mobilidade:** habilidade para transferir programas e executá-los sobre várias plataformas;
- **Segurança:** habilidade para restringir acessos aos recursos;
- **Eficiência:** segurança e mobilidade sem comprometer o desempenho da rede, pelos menos na maioria dos casos;

2.5. Arquitetura do Nodo ativo

A arquitetura de um nodo ativo, de modo geral, deve possuir um ambiente de execução transiente. É neste ambiente que os programas são executados e os dados processados.

Além do ambiente transiente, uma outra questão importante é o acesso aos objetos, tais como tabelas de roteamento que estão além do ambiente volátil.

Tennenhouse [TEN 96] sugere três modos pelo qual o programa pode alcançar os objetos da rede, além do ambiente transiente.

- Componentes fundamentais
- Armazenagem ativa
- Extensibilidade

2.6. Componentes fundamentais

Implementam métodos externos que fornecem acesso controlado aos recursos fora do ambiente de execução transiente [TEN 96].

2.7. Armazenagem ativa

É a capacidade de armazenamento de informações no ambiente de armazenagem persistente, situado no próprio nodo ativo.

2.8. Extensibilidade

Consiste na permissão de programas definirem novos métodos e novas classes [TEN 96]. Estas novas definições podem ser carregadas por demanda, quando necessárias para realizar alguma função sobre os pacotes que chegam no nodo.

A Fig. 2 ilustra a organização de um nodo ativo proposto em [TEN 96].

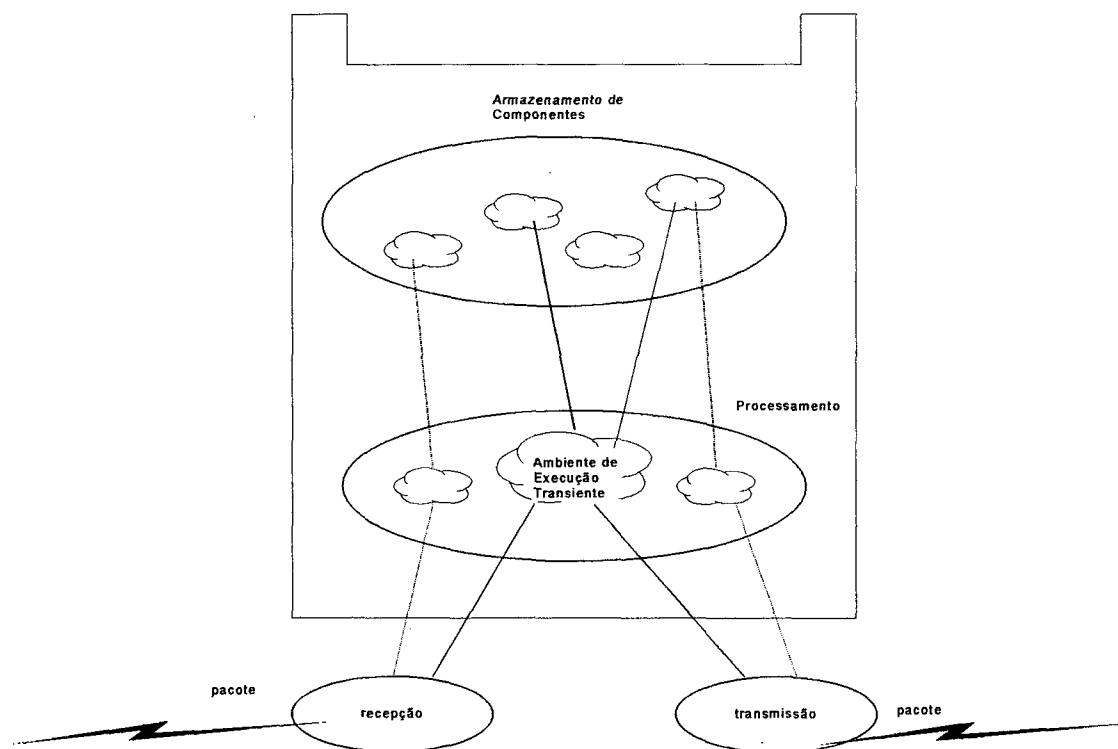


FIGURA 2: ORGANIZAÇÃO DE UM NODO ATIVO

Para analisar as propriedades e os desafios que a rede ativa impõe, em [KUL 97] é descrito um modelo de nodo ativo. Neste trabalho implementa-se um protótipo em *JAVAtm* [JAVA 00]. A linguagem *Java* foi escolhida, por fornecer a máquina virtual *java*, a serialização e a portabilidade entre plataformas.

Em [KUL 97] é descrito uma rede ativa virtual como sendo uma coleção de nodos ativos individuais e um servidor de informações, onde cada um roda sobre uma máquina virtual *Java*. Os *links* virtuais são simulados utilizando-se *sockets* em conexões *tcp*.

A Fig. 3 ilustra um exemplo de rede ativa virtual.

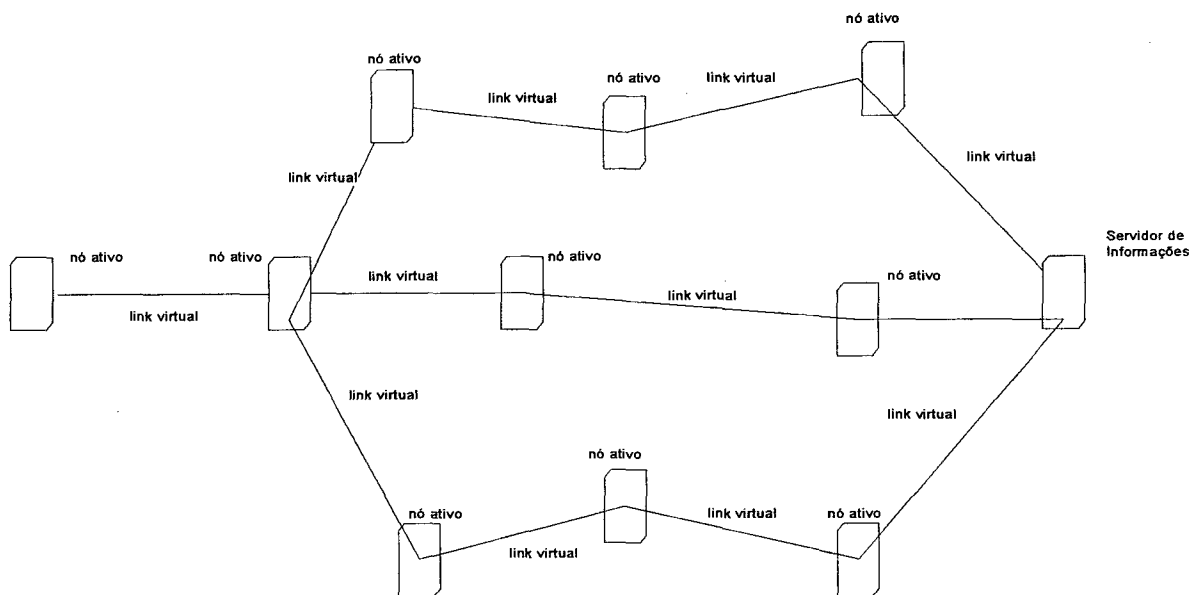


FIGURA 3: EXEMPLO DE UMA TOPOLOGIA PARA REDE ATIVA VIRTUAL

Na arquitetura do nodo, *Kulkarni* define cinco gerentes que tem tarefas específicas: o gerente do nodo, o gerente da porta, o gerente do recurso, o gerente do roteamento e o gerente de *small state*³.

³ *Small state*: dispositivo de armazenagem de informações compartilhadas pelas cápsulas que visitam o nodo.

O gerente do nodo é responsável por iniciar todos os outros objetos de gerenciamento; porta, recurso, roteamento e *small state*. Esses objetos possuem gerentes que ficam responsáveis pelo gerenciamento das portas do nodo, dos recursos do nodo, do roteamento e do *small state*, respectivamente.

O gerente da porta envia uma mensagem ao gerente de roteamento indicando que a porta está pronta e que esse gerente inclua a nova porta na tabela de roteamento. Então o gerente da porta fica em *loop* lendo as cápsulas que chegam da rede e enviando os pacotes que saem. O gerente de recursos oferece uma *interface* para os recursos do nodo. Alguns dos recursos básicos são CPU, memória e entrada/saída. O gerente do *small state* é a abstração utilizada para armazenamento no *small state*. O gerente de roteamento mantém as tabelas de roteamento e fornece a *interface* para manipulação destas tabelas. Na Fig. 4 é mostrado a cápsula que flui na rede.

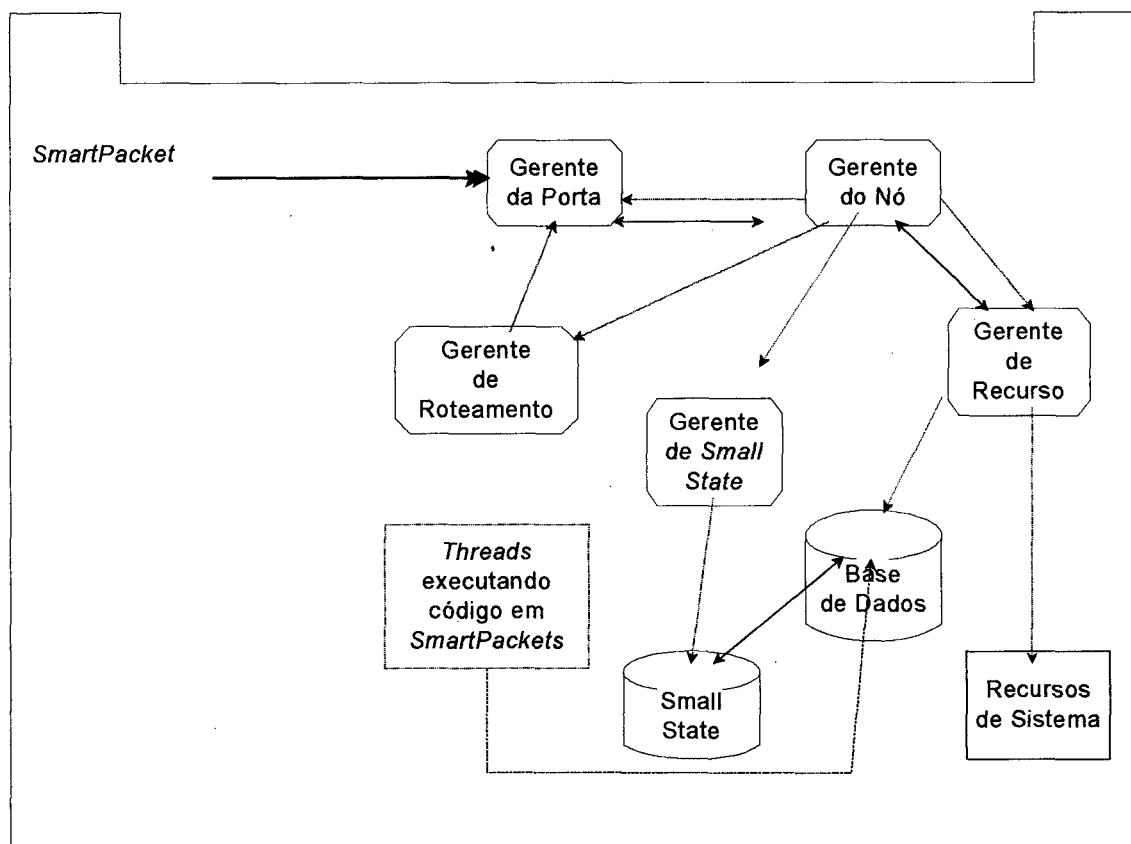


FIGURA 4: MODELO DE NODO ATIVO PROPOSTO POR A. B. KULKARNI

Essa cápsula é denominada de *SmartPacket*, o qual representa o código e os dados transportados em formato de *bytecode* gerado pelo compilador *Java*.

Esse formato é ilustrado na Fig. 5 que apresenta o formato do *SmartPacket*. Quando o *SmartPacket* chega ao nodo ativo, este utiliza a definição da classe para defini-la no ambiente de execução [KUL 97]. Então, o objeto é *deserializado*, iniciando a instância da classe. A instância é então executada em sua própria *thread*.

TypeID	Nome Classe	Tamanho Classe	Definição Classe	Tamanho Objeto	Objeto Serializado
--------	-------------	----------------	------------------	----------------	--------------------

FIGURA 5: FORMATO DO *SMARTPACKET*

2.9. Benefícios das Redes Ativas

Em [TEN 96] está descrito as três principais vantagens de uma arquitetura de redes ativas sobre o atual modelo de transporte passivo de *bits*, quais sejam:

- **Transferência de código:** que fornece a base para os protocolos adaptativos habilitando-os às interações mais ricas do que a transferência de formatos de dados fixos;
- **Conceito de Cápsula:** que dá suporte na implementação de funções para aplicações específicas em pontos estratégicos dentro da rede;
- **A abstração da programação:** que fornece uma plataforma poderosa para personalização da infraestrutura dirigida ao usuário, permitindo novos serviços serem distribuídos de forma mais rápida.

Contudo, [DEL 99] relata que a evolução do atual modelo de rede para redes ativas dependerá dos benefícios que possam ser obtidos pelas aplicações. Para [DEL 99] esses benefícios podem ser categorizados da seguinte forma:

- Disponibilidade de informações seguras pelos nodos intermediários;
- Capacidade de processamento de dados durante o caminho;
- Adoção de estratégias distribuídas;

- Facilidade no desenvolvimento de novos serviços de rede;

2.9.1. Disponibilidade de informações seguras pelos nodos intermediários

Com a disponibilidade dos dados nos nodos em uma rede poder-se-á extrair essas informações enviando-as de volta para a aplicação, ou utilizá-las para tomada de decisões autonomamente. Para isso, agentes móveis podem ser encapsulados e transportados dentro das cápsulas e são ideais para a realização deste benefício [VAN 2000].

2.9.2. Capacidade de processamento de dados durante o caminho

A capacidade de processamento de dados nos diversos nodos intermediários de uma rede tem importância fundamental para a implementação de novas aplicações.

Este benefício é realizado pela inserção de funções específicas nos nodos intermediários, e essas têm a capacidade de consultar e modificar os dados endereçados a outros nodos. As modificações que porventura sejam necessárias estarão condicionadas ao corrente estado da rede ou às necessidades do nodo receptor. Logo, poder-se-á adaptar o fluxo de dados para diferentes requisitos de largura de banda e os congestionamentos da rede.

2.9.3. Adoção de estratégias distribuídas

A distribuição da computação nos diversos nodos da rede é vital para o aumento do desempenho da rede. *Firewalls* distribuídos [TEN 96] e o gerenciamento distribuído de árvores *multicast* [LI- 98] são alguns dos exemplos que necessitam da distribuição da computação na rede.

2.9.4. Facilidade no desenvolvimento de novos serviços de rede

A injeção de código nos diversos nodos na rede possibilita a criação de novos protocolos, serviços e aplicações. Essas melhorias na rede podem ser realizadas

remotamente, possibilitando que as atualizações sejam rapidamente incorporadas nos dispositivos da rede.

2.10. Considerações finais

O paradigma de Redes Ativas surgiu entre 1995 e 1996 com o objetivo de mudar a forma como as redes são administradas e operacionalizadas. No entanto, o conceito de redes ativas não é totalmente novo, pois diversas aplicações já incorporam esse paradigma. Dentre elas, temos, a filtragem de pacotes em *firewalls*, onde os filtros decidem quais pacotes devem passar e quais devem ser bloqueados. É uma aplicação que considera um nodo intermediário que realiza computações (filtragem) dos dados que fluem de um ponto da rede para outro. Aplicações *WEB proxies*, pesquisas de dados, roteadores *multicast* e *gateways* de vídeo são outros exemplos que absorvem o conceito de redes ativas.

Estas aplicações incorporam o modelo de redes ativas e é neste aspecto que alguns pesquisadores visualizam uma nova área de pesquisa mais realizável ao curto prazo do que aquela proposta por Tennenhouse (Redes Ativas ao Nível de Rede), denominada de **Redes Ativas ao Nível de Aplicação**.

3. CONCEITOS DE REDES ATIVAS NO NÍVEL DE APLICAÇÃO

O paradigma de redes ativas na camada de rede promete solucionar diversos problemas encontrados na atual arquitetura de rede. Contudo, suas pesquisas ainda são embrionárias e precisam ser desenvolvidas tanto no nível acadêmico quanto no nível de mercado/indústria.

Diante deste fato, surge uma visão em maior nível de abstração para o paradigma de redes ativas conhecido como redes ativas no nível de aplicação (RANA); o qual mostra-se mais concretizável ao curto prazo em relação à rede ativa sugerida em [TEN 96]. Para isso, abstraem-se os conceitos de redes ativas para as aplicações desenvolvidas na camada da aplicação do protocolo OSI da ISO.

Um sistema de rede ativa no nível de aplicação é proposto por [FRY 98] que consiste em um ambiente cliente/servidor.

A comunicação entre cliente e servidor é melhorada com servidores de *proxy* dinâmico (SPD) que estão localizados em pontos estratégicos no caminho entre o servidor e o cliente, funcionando como o **nodo ativo** definido em [TEN 96].

3.1. Servidor de *proxy* dinâmico (SPD)

No modelo atual da arquitetura Internet, um pedido feito por um cliente ao servidor de serviços é efetuado via os protocolos http e tcp.

A proposta de *Michael Fry* inclui outros servidores, o SPD que recebe os pedidos dos clientes e carrega(m) o(s) programa(s) específico(s) que assume(m) a tarefa de produzir resposta(s) mais eficiente(s) para os clientes.

O SPD exporta um conjunto de *interfaces* para os pedidos dos clientes, usando RMI [JAV 00]. Os pedidos que estão implementados são:

- **Load**: método de carga de programas específicos;
- **Run**: método de execução de programas específicos;
- **Modify**: método de alteração do estado de programas específicos;
- **Stop**: método de finalização de programas específicos;

3.2. Os programas específicos (proxylets)

Os programas específicos, denominados de *proxylets* em [FRY 98], são programas que atuam como filtros ou funcionalidades que melhoram a *performance* do nível de serviço entre servidores e clientes.

Os *proxylets* são carregados dos servidores de *proxylet* para dentro dos SPDs. “O objetivo de se ter um servidor deste gênero é o compartilhamento de código e a validação de *proxylets* originados de servidores confiáveis” [FRY 98].

Uma das tarefas básicas dos *proxylets* é a comunicação com os servidores WEB⁴, buscando os dados necessários para a elaboração das respostas ao cliente.

3.3. O servidor de protocolo dinâmico

Outro componente que está presente na arquitetura proposta em [FRY 98] é o servidor de protocolo dinâmico, o qual fornece as pilhas de protocolos que podem, ou não, serem carregadas dentro do SPD para interoperabilidade entre *proxylets* e o sistema [FRY 98].

A Fig. 6 exibe a arquitetura RANA proposta em [FRY98].

⁴ Web: Rede de documentos em formato HTML que estão interligados em servidores do mundo inteiro.

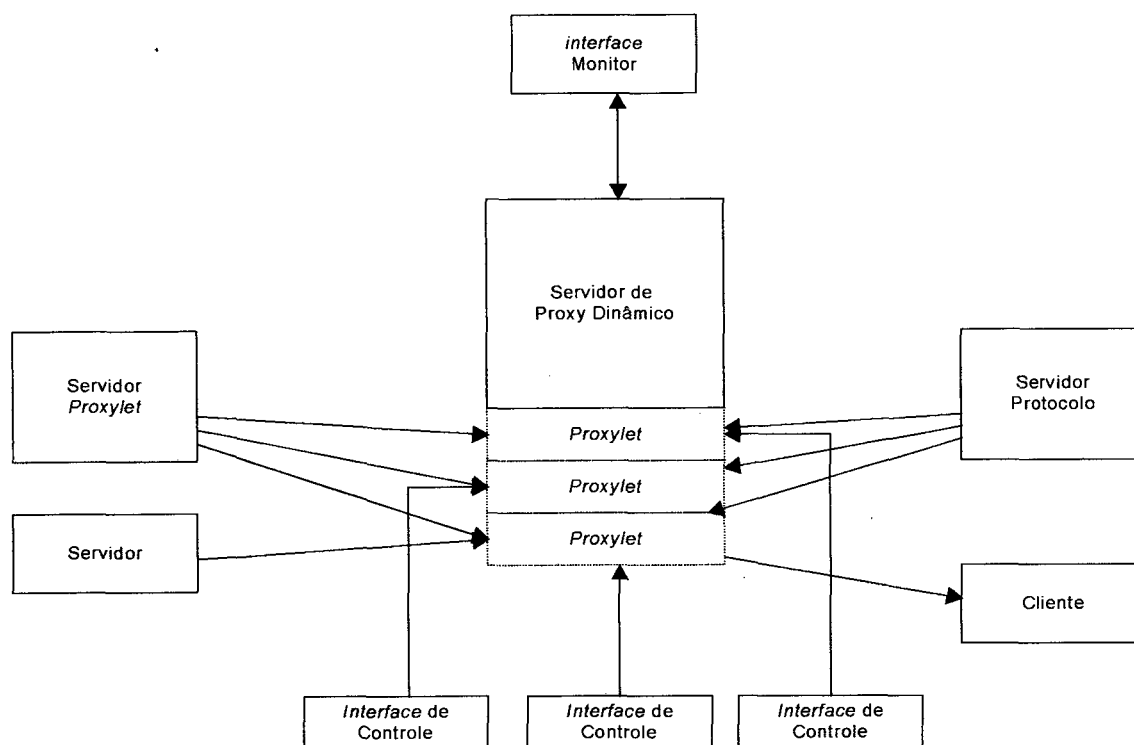


FIGURA 6: VISÃO DA ARQUITETURA RANA

3.4. Aspectos de implementação

A arquitetura descrita em [FRY 98] está implementada inteiramente na linguagem *Java*. Esta linguagem é autojustificável pela interoperabilidade, a *serialização* e a máquina virtual fornecida. Porém, *Fry* salienta que sua arquitetura não é dependente deste ambiente.

3.4.1. Servidor de proxy dinâmico

O SPD espera os pedidos dos clientes e fornece um ambiente para execução de *proxylets*.

Uma referência ao *proxylet* é passada ao SPD na forma de *Universal Resource Locator*⁵ (URL).

O SPD é configurado para usar um *cache WEB* local, tomando vantagem da infra-estrutura que esta tecnologia fornece.

Cada *proxylet* roda em uma máquina virtual Java distinta, por questões de simplificações na implementação [FRY 98].

A invocação de método remoto em RMI [RMI00] é utilizada para a comunicação entre SPD e cliente.

As duas principais *interfaces* exportadas pelos SPDs envolvem a carga, execução e controle dos *proxylets*; e o gerenciamento que é usado para reter o estado dos *proxylets* carregados no SPD.

O método *stop* responsabiliza-se por terminar a execução do *proxylet*, e se após três segundos este não for eliminado, a máquina virtual Java tem a sua execução paralisada.

3.4.2. *Proxylet*

As operações que o *proxylet* efetua são flexíveis o suficiente, permitindo que ele acesse todas as classes do ambiente *Java*.

Fisicamente um *proxylet* é uma coleção de arquivos *.class*⁶ em *Java*, e estes arquivos são empacotados juntos, em um arquivo *Java* de extensão *.jar*,⁷ ou seja, cada um dos *proxylets* consiste de um simples arquivo *jar*. Estes arquivos ficam armazenados em servidores *WEB*. Logo, podemos referenciá-los via *url*.

3.5. A arquitetura de controle do RANA

O núcleo do sistema proposto em [FRY 98] é o SPD, o qual recebe os pedidos dos clientes e carrega os *proxylets* adequados para processar as requisições e gerar as respostas adequadas aos emissores do pedido.

⁵ URL: esquema utilizado na web para localizar uma determinada página ou arquivo.

⁶ Class: extensão dos arquivos gerados pelo compilador *javac* de Java.

⁷ Jar: formato padrão utilizado em *Java* para armazenar arquivos de classe de maneira agrupada.

As localizações dos SPDs são conhecidas e pode haver um ou mais SPDs na comunicação entre cliente e servidor. A comunicação entre cliente e SPD é realizável através de conexões explícitas em RMI.

Os SPDs devem conhecer a existência do outro e a descoberta da localização de cada um deve ser automática. Estratégias de localização mais genéricas são deixadas para trabalhos futuros.

3.6. A *interface* de controle

A *interface* de controle genérica é usada para iniciar o *download* de *proxylets* para o SPD.

Uma operação é definida para passar novos parâmetros aos *proxylets*, enquanto ele está rodando. *Michael Fry* considera que as *interfaces* de controle especializadas serão desenvolvidas para que algum *proxylet* habilite interações com o usuário [FRY 98].

3.7. A *interface* de monitor

Há uma *interface* de monitoramento para observar o estado da execução dos *proxylets*. As informações são repassadas à *interface* de controle para tomar o controle da execução [FRY 98].

3.8. Aspectos de segurança

A comunidade científica de redes ativas ‘pura’ critica as bases de segurança no trabalho de [FRY 98].

Realmente, no trabalho descrito em [FRY 98] não foi desenvolvido um modelo de segurança adequado. Ele próprio relata que este aspecto ainda será alvo de trabalhos futuros.

Contudo, no sentido de amenizar o problema de segurança, [FRY 98] restringiu os servidores de *proxylets* e os servidores de *proxy* dinâmico a um domínio de um

simples proprietário. Desta forma, os SPDs não permitem a execução de *proxylets* de fontes não confiáveis.

Na implementação corrente em [FRY 98], os seguintes aspectos de segurança são colocados:

- Somente *hosts* confiáveis possuem permissão para submeter pedidos ao SPD;
- *Proxylets* são unicamente carregados de servidores WWW confiáveis;
- São fornecidas autenticações e assinaturas digitais aos *proxylets*, por efetuarem operações não confiáveis dentro do ambiente do SPD.

Neste caso, a linguagem *Java* também possui mecanismos de segurança que impedem que os *proxylets* acessem dados não autorizados sobre a máquina que o SPD está executando. Logo, o trabalho em [FRY 98] fica unicamente dependente dos aspectos de segurança fornecidos pela ambiente *Java*, pelo menos, até que seja implementado um modelo de segurança para a sua arquitetura.

4. XML COMO UMA TECNOLOGIA PARA ABSTRAÇÃO DOS CONCEITOS DE REDES ATIVAS NO NÍVEL DE APLICAÇÃO

A linguagem XML desponta como uma saída alternativa para a linguagem HTML. Não apenas na área comercial, mas em todas as áreas, as capacidades de extensão da linguagem XML a torna a linguagem ideal quando se trata de representar informações estruturadas.

A seguir, temos uma introdução sobre as características gerais da XML. No desenvolvimento deste subcapítulo houve a preocupação de dar uma visão geral da linguagem como uma pequena introdução a todos os seus aspectos.

4.1. Introdução a XML

O final da década de 90 foi marcado por um rápido e expressivo crescimento da *Internet*. A grande rede mundial de computadores tornou-se notícia e começou a receber o acesso de grandes e pequenas empresas, além de pessoas físicas.

O padrão adotado pela *Internet* para o intercâmbio de dados foi a HTML – *HyperText Markup Language* – Linguagem de Marcação de Hipertexto. Este padrão foi criado baseado no SGML – *Standard Generalized Markup Language* – Linguagem Padrão de Marcações Genéricas.

SGML foi um modelo criado em 1986 (ISO 8879) para ser um padrão de marcação generalizada. De acordo com *Richard Light*, a SGML é um padrão “muito usado para codificar documentos estruturados, variando em tamanho e complexidade.” [LIG99]

São exemplos de aplicação do padrão SGML a linguagem *Edgar* (que fornece uma marcação para relatórios financeiros para repartições públicas), o PCIS (que é um

padrão de marcação para armazenamento de dados de semicondutores) e a própria linguagem HTML. Ver Fig. 7.

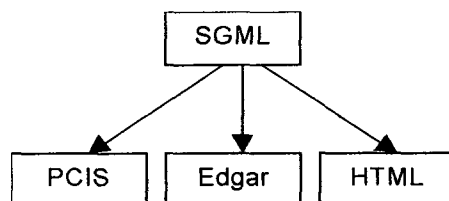


FIGURA 7: RELAÇÃO ENTRE SGML, PCIS, EDGAR E HTML

O problema da SGML é que ela fornece um padrão muito complicado para ser usado na Internet. Para isto, criou-se a HTML que é uma linguagem mais simples, baseado no SGML, próprio para ser usado na rede mundial. A marcação HTML é, uma aplicação da marcação SGML.

A HTML fornece uma maneira fácil, rápida e prática de se formatar texto para a *Internet*. O problema é que, com o rápido crescimento deste veículo de comunicação e com a entrada de grandes empresas (e, principalmente, com o grande crescimento do comércio eletrônico), a HTML já não serve para os propósitos desejados. Poderia-se pensar em utilizar a SGML em seu lugar, mas esta linguagem ainda é considerada de muita complexidade. Em seu lugar, desenvolveram uma versão simplificada deste padrão, a linguagem XML. De acordo com *Richard Light*:

O objetivo era fornecer aos usuários da WEB maneiras de identificar seus próprios comandos (*tags*) e atributos quando desejassem, em vez de ter de usar o esquema de comandos da HTML. Em novembro de 1996, o grupo de trabalho do *World Wide WEB Consortium* SGML estava pronto para anunciar o primeiro esboço da XML - ou *Extensible Markup Language* - na conferência de SGML em Boston, EUA.

[LIG99]

4.2. Características de XML

Conforme a citação, a sigla XML significa *eXtensible Markup Language*, ou Linguagem de Marcação Extensível. Isto quer dizer que a XML, ao contrário da HTML,

permite aos usuários definir suas próprias *tags* de marcação de texto. Portanto, enquanto num documento HTML pode-se apenas definir a formatação dos elementos, num documento XML pode-se definir o que estes dados significam, agregando mais informação a um documento.

A XML permite aos usuários da *Internet* criar dados estruturados e definir informações sobre a estrutura destes dados. Desta forma, os dados tornam-se informações aos olhos de um usuário. O quadro 1 mostra trechos de documento em HTML e XML.

```

<!-- Trecho de documento HTML -->

<h1>Invoice</h1>
<p>From: Joe Bloggs
<p>To: A. Another
<p>Date: 1 Feb 1999
<p>Amount: $100.00
<p>Tax: 21%
<p>Total Due: $121.00

<!-- Trecho de documento XML -->

<Invoice>
<From>Joe Bloggs</From>
<To>A. Another</To>
<Date year = '1999' month = '2' day = '1' />
<Amount currency = 'Dollars'>100.00</Amount>
<TaxRate>21</TaxRate>
<TotalDue currency = 'Dollars'>121.00</TotalDue>
</Invoice>

```

QUADRO 1: TRECHOS DE DOCUMENTOS HTML E XML

Comparando os dois trechos de documentos HTML e XML acima, pode-se notar que a linguagem XML é auto-explicativa, isto é, sua estrutura revela mais a respeito do significado do documento do que a linguagem HTML. Esta última apenas preocupa-se em formatar os dados para uma visualização final mais agradável.

4.3 Vantagens de se definir as próprias *tags*

A possibilidade de se definir as próprias *tags* abre infinitas possibilidades para a publicação na WEB. Uma página HTML carrega consigo, somente, informações sobre parágrafos, cores, fontes. Ver Fig. 8.

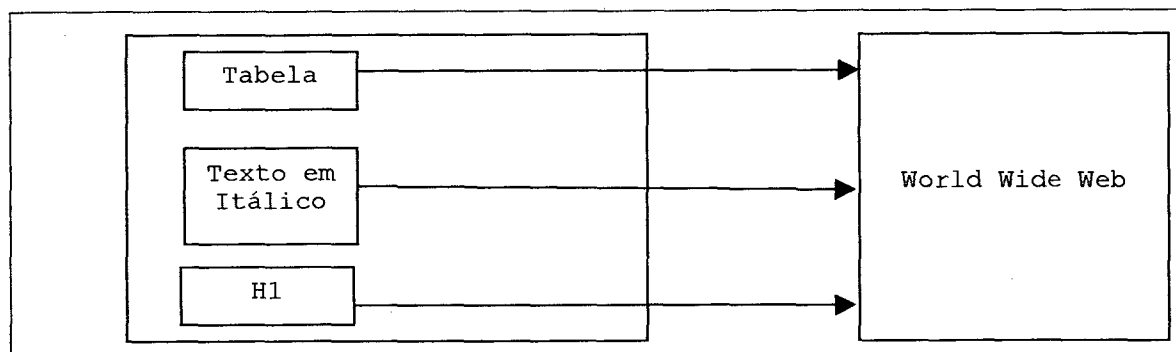


FIGURA 8: DOCUMENTO HTML SENDO EXIBIDO NA WEB

Segundo *McGrath*: “Como consequência, quando este documento se torna disponível *na World Wide WEB*, os diferentes utilitários de pesquisa e usuários vêem somente um conjunto de níveis, tabelas, texto em itálico, etc.” [McG99]

Enquanto isso, uma página XML carrega informações sobre os dados que esta contém. Ver Fig. 9.

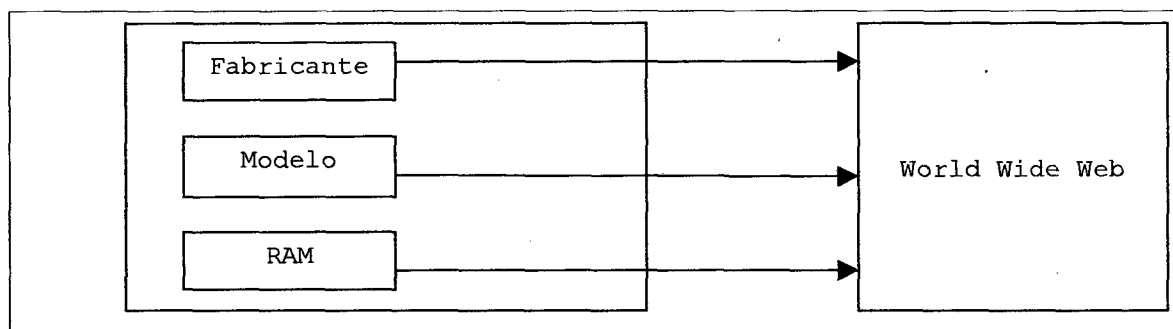


FIGURA 9: DOCUMENTO XML SENDO EXIBIDO NA WEB

Sean McGrath descreve algumas vantagens da representação no formato XML:

- Permitir ao navegador fazer o trabalho de formatação dos dados na tela do usuário. Talvez permitindo aos usuários escolher entre uma variedade de 'aparências' ou formatos de apresentação para os mesmos dados.

- Permitir ao navegador do usuário realizar cálculos a partir dos dados, e manipular e exibir os resultados em várias formas.

- Possibilitar a pesquisa inteligente das informações, por exemplo: 'Encontrar todos os PCs à venda na World Wide Web com capacidade de disco rígido superior a 2 GB'.

- Verificar de forma inteligente se todas as informações necessárias relativas a uma entrada apropriada na página da Web de vendas de PC estão realmente lá, por exemplo: 'Todos os PCs devem conter um elemento de tamanho de RAM e podem conter opcionalmente um elemento de tamanho de disco rígido.' [McG99]

Segundo *McGrath*: "A filosofia da essência da XML apareceu como resultado de uma análise longa e cuidadosa do que realmente significa o termo 'documento' no mundo digital." [McG99]

O termo documento compreende três itens distintos – conteúdo, apresentação e estrutura. Muitas vantagens são obtidas, ao separar estes três itens. Podemos obter apresentações diferentes dos mesmos dados para o contador e para o engenheiro da empresa. Podemos alterar os dados de várias apresentações, simplesmente, alterando um só arquivo. Por outro lado, podemos estender a estrutura possibilitando representar uma quantidade maior de dados.

4.4. Documento de Tipo de Dados (DTD)

Os documentos XML definem uma estrutura dos dados a serem exibidos. Esta estrutura também pode ser validada através de um documento separado denominado DTD – *Data Type Document* – ou Documento de Tipos de Dados. Este documento define as estruturas de dados que deverão ser utilizadas dentro do documento XML. Neste caso, o programa interpretador XML fará uma análise do XML com base nas estruturas definidas no DTD e emitirá uma mensagem de erro caso encontre alguma inconsistência. Neste ponto, nota-se uma diferença básica entre os formatos HTML e XML. Enquanto o primeiro dificilmente emite mensagens de erro, o último é mais

rígido quanto à estrutura do documento. Isto se torna uma vantagem à medida que, em documentos grandes, o número de erros involuntários tende a crescer. Assim, uma linguagem que auxilie na correção destes erros vem a ser de grande ajuda.

A validação supracitada através do uso de um DTD é opcional. Assim, define-se o conceito de documentos válidos contra o conceito de documentos bem-formatados. Estes últimos não são submetidos a um DTD. Neste caso, basta que estes documentos tenham seus elementos internos (também chamados de *tags*) bem aninhados formando uma estrutura de árvore. Já os documentos válidos são aqueles que foram validados através de um DTD.

O fato de se poder utilizar documentos sem validação (apenas bem formados), pode gerar desconfianças. Porém isto não acontece, conforme analisa *Richard Light*:

O conceito de ser bem formado soa como um convite à anarquia, mas esse não é o objetivo. O conceito é fornecido em XML porque, por muitos motivos, não é necessário fornecer uma DTD ou formalmente verificar um documento quanto àquela DTD. Ler uma DTD ocupa tempo e espaço; verificar um documento XML quanto a DTD ocupa ainda mais tempo. Se um servidor de documentos XML já se assegurou que todos são válidos e limpos, é uma perda de tempo que cada cliente tenha de fazer o mesmo. [LIG99]

4.5. Formatação

Assim como a definição dos tipos usados no documento é separada do documento em si, a definição da formatação dos elementos também é separada. A XML define uma linguagem de estilo para ser usada na formatação de seus documentos, denominada XSL (*XML Style Language* – Linguagem de Estilo da XML). Ver fig. 10. "Da mesma forma que a XML é um subconjunto do Padrão Internacional da SGML (ISO 8879), a XSL é um subconjunto simplificado da linguagem de estilo padrão Internacional conhecida como DSSSL (ISO/IEC 10179)." [McG99]

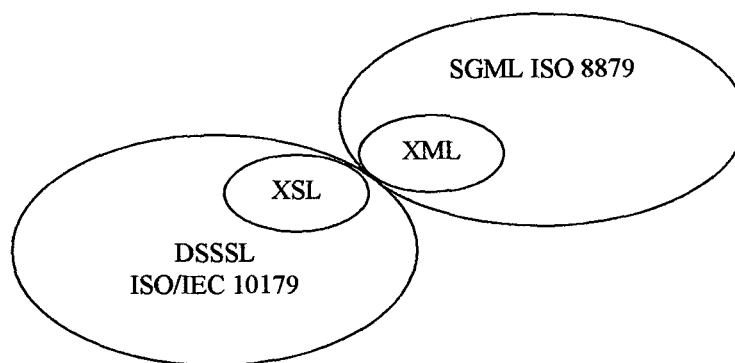


FIGURA 10: A RELAÇÃO ENTRE PADRÕES DE ESTILO E DOCUMENTAÇÃO

A base dos documentos XML é a separação dos dados (XML), da formatação (XSL) e da estrutura dos dados (DTD). Juntando estes três itens, temos o que constitui um documento XML, de uma forma completa. Ver fig. 11.

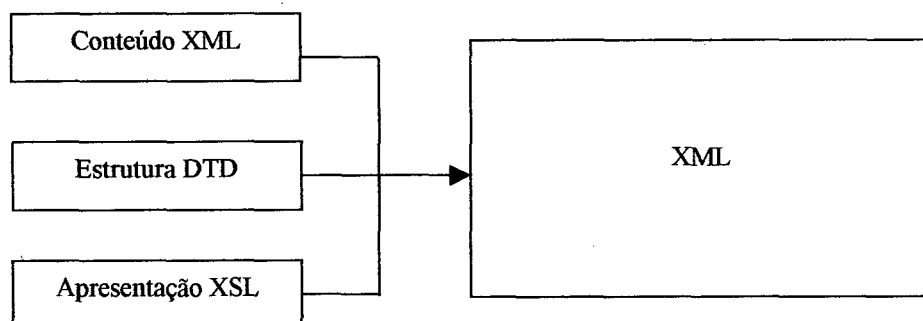


FIGURA 11: CONTEÚDO, ESTRUTURA E APRESENTAÇÃO INTEGRADOS PARA EXIBIÇÃO NA *WEB*

4.6. XML na prática

A linguagem XML possui algumas regras para escrita de documentos. Tais regras é que tornam um documento XML mais estruturado. Segue-se abaixo quais seriam estas regras. A seguir, descrever-se-á a criação de um documento XML como exemplo.

4.6.1. Documentos válidos e bem-formatados

A única exigência que deve ser satisfeita para que um documento seja um arquivo XML é a de que ele deve manter uma estrutura perfeita em forma de árvore. A um documento XML que mantenha esta perfeita estrutura em forma de árvore chamamos de documento “bem-formado”. “Basicamente, um documento XML bem formado é um documento a partir do qual o Processador XML⁸ pode criar, com êxito, uma estrutura em árvore.” [McG99]

Um documento XML contém, portanto, uma estrutura lógica e uma estrutura física. A estrutura lógica compreende a estrutura em forma de árvore, do documento XML. No topo desta estrutura está o elemento raiz, a partir do qual todos os outros elementos derivam. Ver Fig. 12.

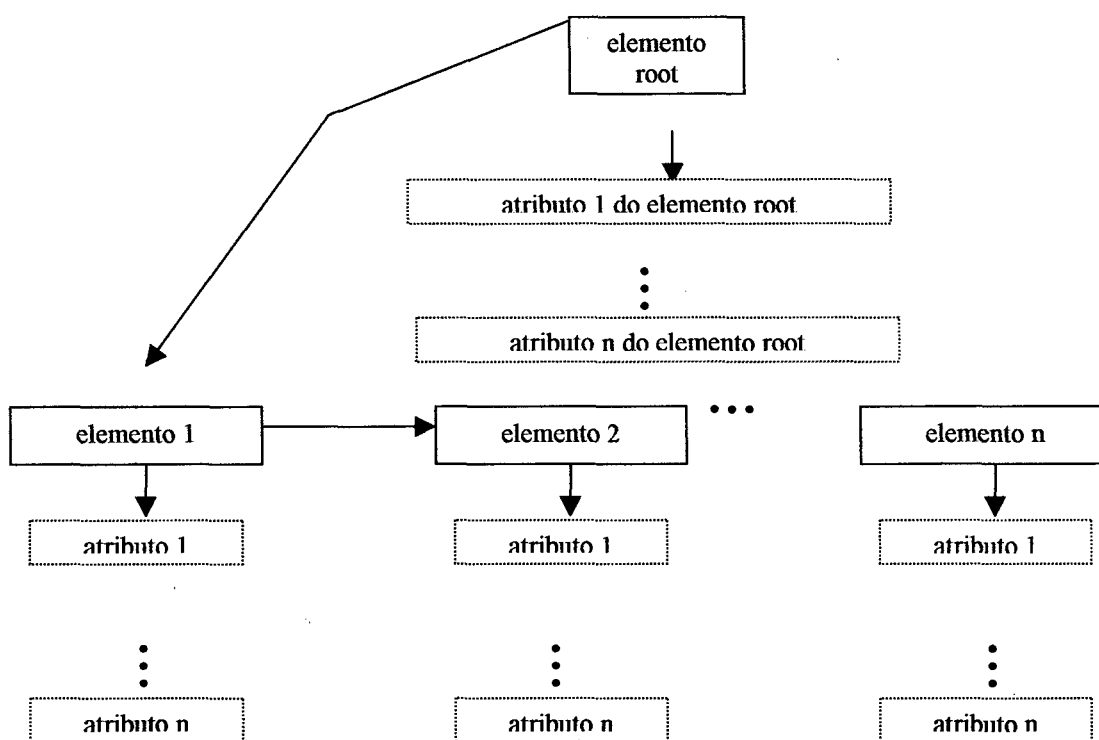


FIGURA 12: ESTRUTURA LÓGICA DE UM ARQUIVO XML

⁸ Por processador XML entenda-se quaisquer softwares que leiam um documento XML e executem algum tipo de processamento sobre estes dados. Um exemplo de processador XML é o Microsoft Internet Explorer.

A estrutura física de um documento XML compreende a maneira como ele está armazenado na memória. Ver Fig. 13.

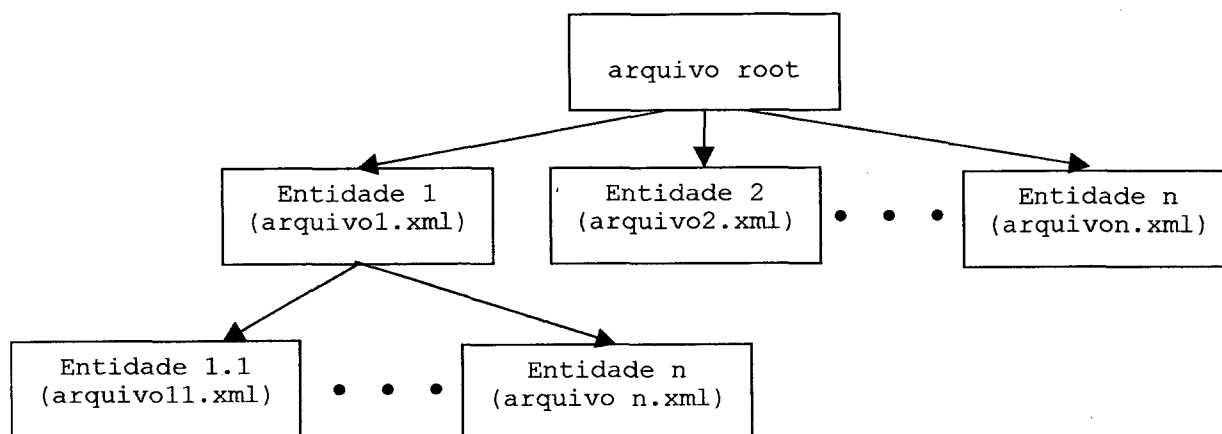


FIGURA 13: ESTRUTURA FÍSICA DE UM DOCUMENTO XML

4.6.2. Regras de formação XML

Documentos XML são documentos estruturados. Por isso, XML tem a sintaxe mais rígida do que HTML. Na violação da sintaxe, o programa processador de XML (neste caso, inclui-se o Microsoft Internet Explorer versão 5.0 ou superior) emitirá uma mensagem de erro.

Abaixo estão descritas as regras de formação para arquivos XML. Todos os arquivos XML devem seguir essas regras. Note que é feita uma comparação com HTML. No entanto, é bom frisar que os elementos XML nada têm a ver com os elementos HTML, ou seja, não existe um elemento para parágrafo, outro para negrito. O usuário é quem define as *tags*. Mais adiante, construir-se-á um arquivo XML.

1ª.Regra:

Todas as *tags* precisam ser fechadas. Ver tabela 1.

TABELA 1: COMPARAÇÃO ENTRE CÓDIGO HTML E CÓDIGO XML QUE MOSTRA COMO XML É MAIS FORMAL

HTML	XML
------	-----

<P>Parágrafo HTML <P>Outro parágrafo HTML	<P>Parágrafo XML </P> <P>Outro parágrafo XML</P>
--	---

Se não for necessário colocar nenhum elemento entre as *tags*, escreve-se, apenas, uma *tag* e coloca-se uma barra no final. Como por exemplo:

2ª.Regra:

As *tag's* não podem ser entrelaçadas, isto é, se você abriu duas *tag's*, aquela que for aberta por último deve ser fechada primeiro. Ver tabela 2.

TABELA 2: COMPARAÇÃO ENTRE CÓDIGO HTML E XML. ANINHAMENTO DOS ELEMENTOS XML EM FORMA DE ÁRVORE

HTML	XML
Olá<I>MeusIrmãos</I>	Olá<I>MeusIrmãos</I>

3ª.Regra:

Manter consistência quanto a caixa (alta ou baixa) de uma *tag*. Não é necessário escrever sempre em maiúscula, mas se você iniciou uma *tag* com letras maiúsculas, deve terminar com maiúsculas. Ver tabela 3.

TABELA 3: COMPARAÇÃO ENTRE CÓDIGO HTML E XML. É SUGERIDO QUE AS TAG'S XML SÃO SEMPRE MAIÚSCULAS OU MINÚSCULAS

HTML	XML
<I>Olá Meus Irmãos</I>	<I>Olá Meus Irmãos</I>

4ª.Regra:

Coloque todos os atributos entre aspas. Ver tabela 4.

TABELA 4: COMPARAÇÃO ENTRE A LINGUAGEM HTML E A LINGUAGEM XML. ATRIBUTOS EM XML ENTRE ASPAS

HTML	XML
	

5ª.Regra:

Use um único elemento raiz (ou *root*). Ver tabela 5.

TABELA 5: COMPARAÇÃO ENTRE HTML E XML. HÁ APENAS UM ELEMENTO PRINCIPAL EM XML

HTML	XML
<pre><TITLE> Marcações desleixadas </TITLE> <BODY> Este arquivo não está bem formado </BODY></pre>	<pre><HTML> <TITLE> Marcações desleixadas </TITLE> <BODY> Este arquivo não está bem formado </BODY> </HTML></pre>

6ª.Regra:

A linguagem HTML define algumas entidades que são usadas para a internacionalização da linguagem. Um exemplo do uso destas entidades é o uso de caracteres acentuados no português. As entidades são delimitadas por um & (e comercial) e um ; (ponto-e-vírgula). XML também define entidades, porém, define apenas o mínimo delas. Veja exemplo:

```
&lt; (<)
&gt; (>)
&amp; (&)
```

7ª.Regra

Scripts em HTML podem conter caracteres especiais como < e &. Por este motivo é aconselhável colocar estes blocos de código em seções *CDATA* para que elas não sejam processadas. As seções *CDATA* servem justamente para isto: impedir o processamento do código em seu interior. Ver tabela 6.

TABELA 6: COMPARAÇÃO ENTRE LINGUAGEM HTML E XML. EM XML SCRIPTS SÃO ESCRITOS EM BLOCO *CDATA* PARA NÃO ATRAPALHAR O PROCESSAMENTO DOS COMANDOS XML

HTML	XML
<pre><SCRIPT> function less_than_seven(n){ return n<7; } </SCRIPT></pre>	<pre><SCRIPT><![CDATA[function less_than_seven(n){ return n<7; }]]></SCRIPT></pre>

4.6.3. Escrevendo uma aplicação

As ferramentas existentes atualmente permitem uma ampla gama de aplicações que podem ser desenvolvidas com o uso desta tecnologia. Para uma melhor compreensão do assunto, construir-se-á uma aplicação XML que exemplifica as etapas de construção de um documento XML.

Para isto considere o seguinte problema: deseja-se construir uma página que trate da terapia médica da medicina alternativa, a homeopatia. Nesta página, deseja-se colocar uma lista de plantas medicinais ligadas a moléstias que podem ser curadas com estas plantas. Pode-se resolver este problema seguindo as etapas abaixo relacionadas:

1ª. Etapa: Construção de um DTD

Esta etapa é opcional, tendo em vista que as ferramentas utilizadas para construção e visualização de um documento XML não exigem que exista um DTD. No escopo do problema, também não há necessidade de construção de um DTD. Porém, para mostrar o uso desta capacidade do XML, proceder-se-á a construção do DTD.

Seria interessante construir DTD's para testar cada arquivo XML no caso de uma intranet, em que a estrutura dos documentos XML deve ser padrão, ou seja, todas as *tags* devem ser definidas para facilitar a busca e impedir a duplicidade de dados.

Para construir um DTD é necessário que haja inicialmente uma visão de como seria a estrutura do arquivo XML final. Considera-se, então, a estrutura exibida no quadro 2.

```

<root>
  <planta nome="..">
    <molestia nome=".." />
    ..
    <molestia nome=".." />
  </planta>
  ..
  <planta>
  ..
  </planta>
</root>

```

QUADRO 2: MODELO DA ESTRUTURA DO ARQUIVO XML

Assim, tem-se um elemento “root” (que seria o elemento raiz, o que é obrigatório na linguagem XML). Abaixo deste elemento “root”, tem-se vários elementos “planta” (tantos quantos fossem necessários). Esse elemento “planta” tem um atributo “nome” que contém o nome da planta. Abaixo desse elemento “planta”, têm-se vários elementos “molestia” (da mesma forma, tantos quantos fossem necessários) que possui, também, um atributo “nome” que serve para designar o nome da moléstia. Note que se têm as moléstias todas agrupadas pela planta que as cura, numa estrutura perfeita.

Veja o DTD construído para validar esta estrutura no quadro 3.

```
<!ELEMENT root (planta*)>
<!ELEMENT planta (molestia*)>
<!ATTLIST planta nome CDATA " ">
<!ELEMENT molestia EMPTY>
<!ATTLIST molestia nome CDATA " ">
```

QUADRO 3: DTD DO ARQUIVO DE PLANTAS

Através desta linguagem, definimos todos os elementos que constituirão o documento XML, da maior estrutura para a menor. Assim começamos por definir o elemento “root”. Este elemento conterà 0 (zero) ou mais elementos “planta” (quantos forem necessários). Passamos, então, a definir o elemento “planta”, da mesma forma, contendo 0 (zero) ou mais elementos “molestia”. Neste ponto, passamos a definir a lista de atributos que o elemento “planta” conterà. Definimos o atributo “nome” que conterà dados do tipo caractere e iniciará com o valor “ ” (vazio).

Depois, definimos o elemento “molestia”, especificando que deverá ser vazio (não conterà nenhum elemento interno ou texto) e seu atributo “nome” da mesma maneira que o atributo “nome” do elemento “planta”.

Gravamos, então, o nosso DTD num arquivo denominado “plantas.dtd”. Após construirmos nosso arquivo XML poderemos testá-lo utilizando um software específico que aplicará este DTD ao arquivo XML.

2ª. Etapa: Construção do arquivo XML

Esta é a única etapa obrigatória de todo projeto. Construimos um arquivo “plantas.XML” seguindo a estrutura definida pelo DTD. Veja este arquivo no quadro 4.

```
<?XML version="1.0"?>
<!DOCTYPE root SYSTEM "plantas.dtd">
<root>
  <planta nome="ALECRIM">
    <molestia nome="GASES"/>
    <molestia nome="TOSSE"/>
    <molestia nome="ASMA"/>
    <molestia nome="DOR DE CABECA"/>
    <molestia nome="COLICAS MENSTRUAIS"/>
  </planta>
  <planta nome="BERINGELA">
    <molestia nome="DIMINUI TEOR DE COLESTEROL"/>
    <molestia nome="ESTIMULA FUNCOES HEPATICAS"/>
  </planta>
  <planta nome="CANELA">
    <molestia nome="ELEVA A PRESSAO"/>
    <molestia nome="ESTIMULANTE"/>
    <molestia nome="DIGESTIVA"/>
  </planta>
  <planta nome="ATERMISIA">
    <molestia nome="SISTEMA NERVOSO"/>
    <molestia nome="ANEMIA"/>
    <molestia nome="FRAQUEZA"/>
  </planta>
</root>
```

QUADRO 4: ARQUIVO PLANTAS.XML

Seguindo as exigências da ferramenta que utilizaremos (Internet Explorer 5.0), inserimos a linha `<?XML-version="1.0"?>` que especifica que este é um arquivo XML.

Logo abaixo, inserimos a linha `<!DOCTYPE root SYSTEM "plantas.dtd">` que liga o arquivo XML ao DTD construído.

Neste ponto, já podemos visualizar o resultado deste documento através do Internet Explorer 5.0. Abrindo este arquivo, veremos uma estrutura de árvore perfeita, que podemos expandir e contrair para visualizar seus elementos. Ver Fig. 14.

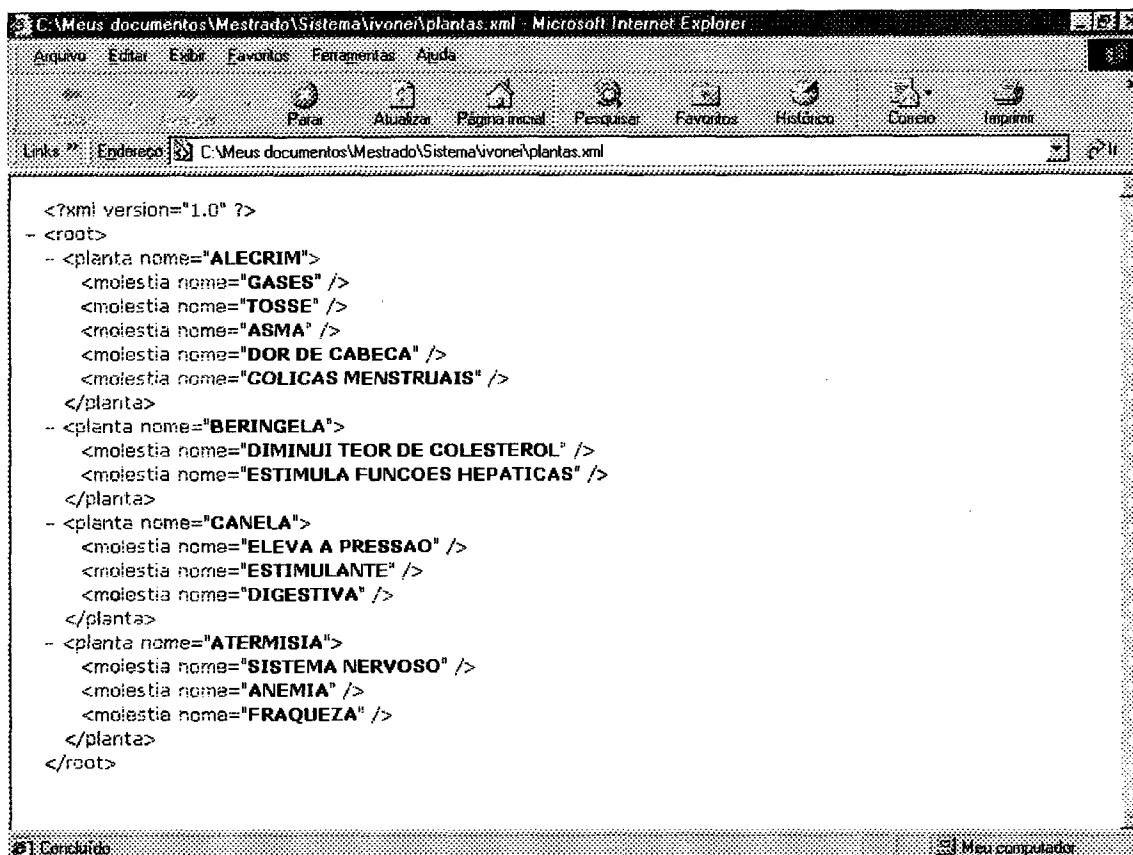


FIGURA 14: ARQUIVO PLANTAS.XML ABERTO ATRAVÉS DO MICROSOFT INTERNET EXPLORER 5.0

Pode-se testar o arquivo XML através do DTD construído no tópico anterior. Existem vários *softwares* específicos que fazem o teste do arquivo XML, aplicando as regras definidas pelo DTD. O XMLINT é um deles. Executando “XMLint plantas.XML”, pode-se validar o documento XML. Veja o resultado no quadro 5. Este resultado indica que não existem erros dentro do XML.

```
C:\>XMLint plantas.XML
plantas.XML
```

QUADRO 5: RESULTADO DO TESTE DO ARQUIVO PLANTAS.XML ATRAVÉS DA APLICAÇÃO DE SUA DTD

3ª. Etapa: Construindo um arquivo de estilo

Para visualizar o documento com uma formatação mais elaborada, podemos construir um arquivo de estilo (XSL) e anexá-lo ao XML.

O arquivo de estilo é construído sobre a idéia de *templates* (modelos). Constrói-se um modelo para cada elemento do arquivo XML. A saída será em formato HTML. Usamos todas as *tag's* HTML para criar a visualização final (o que é uma vantagem, pois HTML é uma linguagem bastante conhecida). Veja o arquivo "plantas.xml" no quadro 6.

```
<?XML version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

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

  <xsl:template match="root">
    <HTML>
      <BODY><xsl:apply-templates/></BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="molestia">
    <LI><xsl:value-of select="@nome"/></LI>
  </xsl:template>

  <xsl:template match="text(">
    <xsl:value-of/>
  </xsl:template>

  <xsl:template match="planta">
    <P><B><xsl:value-of select="@nome"/></B></P>
    <UL>
      <xsl:apply-templates/>
    </UL>
  </xsl:template>
</xsl:stylesheet>
```

QUADRO 6: ARQUIVO DE ESTILO PLANTAS.XSL

Neste caso criamos vários *templates* para os vários elementos constantes do arquivo XML (*molestia*, *root*, *planta*). Dentro destes *templates*, aplicamos *tag's* HTML para gerar a formatação e outras *tag's* específicas da linguagem XSL. Não será entrado

em detalhes da implementação do arquivo de estilo pois isto está fora do escopo deste trabalho. Caso haja interesse por parte do leitor em estudar os arquivos de estilo, uma boa referência é o site da Microsoft (<http://msdn.microsoft.com/XML>).

Depois de construir o arquivo de estilo, precisa-se anexá-lo ao arquivo XML. Faz-se isto inserindo uma linha de referência dentro do arquivo XML. Veja o arquivo “plantas.XML” no quadro 7.

```
<?XML version="1.0"?>
<?XML-stylesheet type="text/xsl" href="plantas.xsl" ?>
<!DOCTYPE root SYSTEM "plantas.dtd">
<root>
  <planta nome="ALECRIM">
    <molestia nome="GASES"/>
    <molestia nome="TOSSE"/>
    <molestia nome="ASMA"/>
    <molestia nome="DOR DE CABECA"/>
    <molestia nome="COLICAS MENSTRUAIS"/>
  </planta>
  <planta nome="BERINGELA">
    <molestia nome="DIMINUI TEOR DE COLESTEROL"/>
    <molestia nome="ESTIMULA FUNCOES HEPATICAS"/>
  </planta>
  <planta nome="CANELA">
    <molestia nome="ELEVA A PRESSAO"/>
    <molestia nome="ESTIMULANTE"/>
    <molestia nome="DIGESTIVA"/>
  </planta>
  <planta nome="ATERMISIA">
    <molestia nome="SISTEMA NERVOSO"/>
    <molestia nome="ANEMIA"/>
    <molestia nome="FRAQUEZA"/>
  </planta>
</root>
```

QUADRO 7: NOVO ARQUIVO PLANTAS.XML

O resultado é apresentado através do Internet Explorer (versão 5.0 ou superior) na Fig. 15.

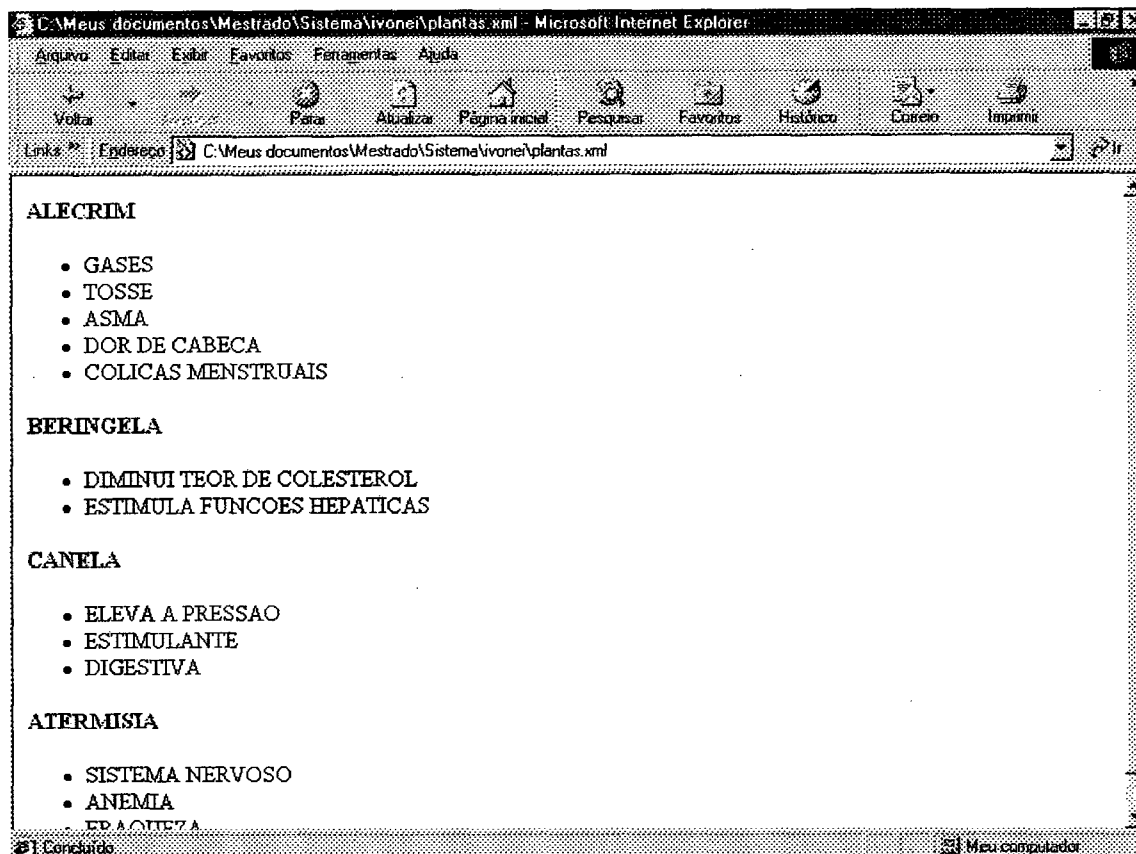


FIGURA 15: ARQUIVO PLANTAS.XML ABERTO ATRAVÉS DO INTERNET EXPLORER 5.0

Note que o Internet Explorer (versão 5.0 ou superior) fez todo o processamento necessário para gerar a saída formatada final, simplesmente porque referenciamos o arquivo XSL dentro do XML.

Porém, o resultado gerado nos mostra as moléstias agrupadas pela planta que as cura, da mesma forma que o arquivo XML. E se tivesse a moléstia e quisesse saber a planta que a cura? O arquivo XSL mostrado no quadro 8 dá a solução.

```
<?XML version="1.0"?>
<xsl:stylesheet XMLns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<BODY>
<TABLE>
<TR>
<TD>
<B>Nome da Molestia</B>
```

```

</TD>
<TD>
  <B>Nome da Planta</B>
</TD>
</TR>
<xsl:for-each select="root/planta/molestia" order-by="@nome">
  <TR>
    <TD>
      <xsl:value-of select="@nome"/>
    </TD>
    <TD>
      <xsl:value-of select="../@nome"/>
    </TD>
  </TR>
</xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

QUADRO 8: NOVO ARQUIVO PLANTAS.XSL GERADO COM O PROPÓSITO DE IMPOR UMA NOVA ORDENAÇÃO AO ARQUIVO PLANTAS.XML

Este arquivo XSL cria uma tabela e insere as plantas agrupadas pelas moléstias (em ordem alfabética). Basta referenciar este arquivo no cabeçalho do arquivo XML (da mesma forma que o exemplo anterior). Pode-se visualizar o resultado na Fig. 16.

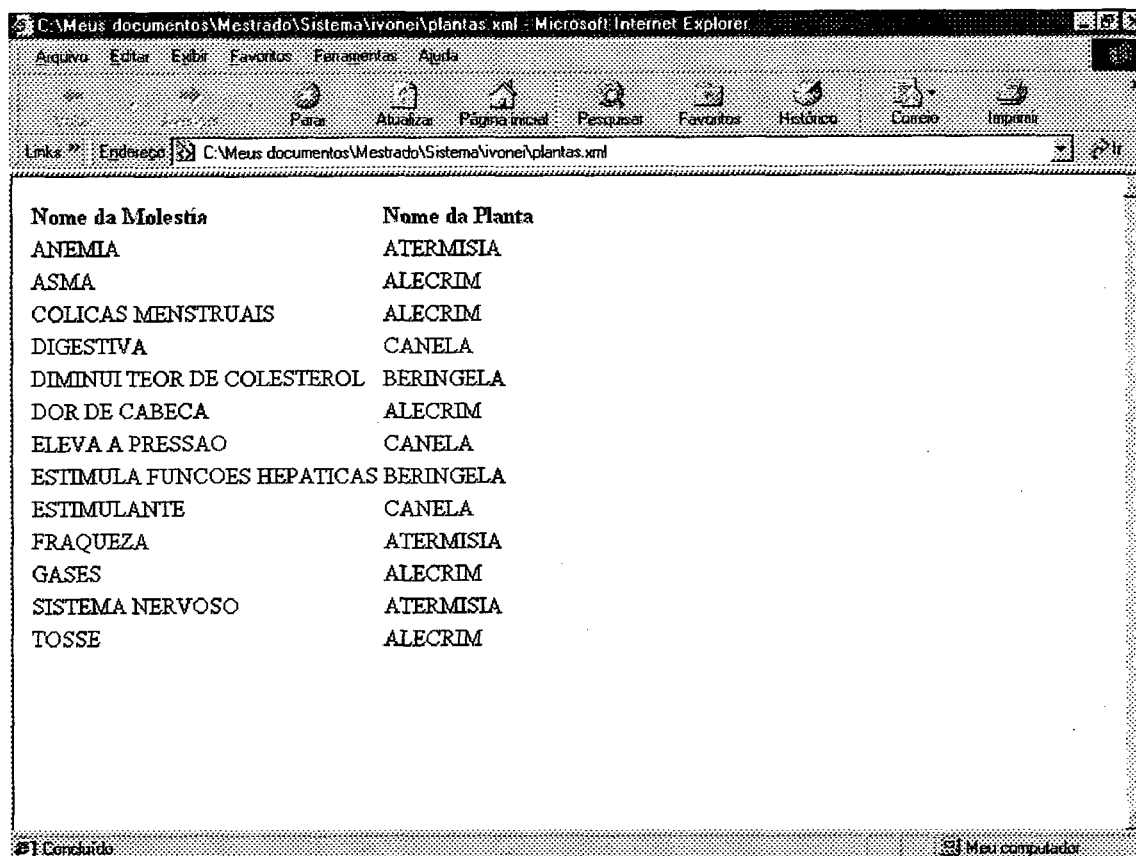


FIGURA 16: ARQUIVO PLANTAS.XML ABERTO ATRAVÉS DO MICROSOFT INTERNET EXPLORER 5.0

Esta é apenas uma das maneiras de se utilizar o XML. A criação de arquivos de estilo só é possível porque o Internet Explorer (versão 5.0) reconhece os comandos utilizados dentro dos arquivos envolvidos. Outras possibilidades abertas pelo XML são analisadas abaixo.

- Existem programas que convertem arquivos XML para HTML através da aplicação de arquivos XSL. Estes programas se tornam úteis quando há possibilidade de se acessar a página construída através de navegadores que não reconhecem o XML (como o Netscape Navigator versão 4.7 ou inferior). Assim, podemos utilizar as capacidades da XML durante o desenvolvimento da página para depois publicá-la em HTML;

- O Microsoft Internet Explorer 5.0 possui a possibilidade de uso de um objeto denominado XMLDSO (*XML Data Source Object*) (ver <http://msdn.microsoft.com/XML>), que é uma interface para um objeto que permite acessar um documento XML como se fosse uma base de dados tradicional. Através deste objeto pode-se incluir elementos dentro do HTML que irão referenciar os dados no arquivo XML;
- Pode-se, também, incluir dados XML dentro de arquivos HTML. Para isto utilizamos a *tag* <XML>. A este recurso damos o nome de *Data Island*;
- Podemos, também, escrever *scripts* em linguagem *JavaScript* ou *VBScript* que acessam dados XML e embutí-los nos arquivos HTML. Isto é possível graças a um padrão para acesso a dados XML chamado DOM (ver McGRATH, 1999). O padrão DOM define um conjunto de objetos que representarão o arquivo XML inteiro. Estes objetos têm propriedades que contêm os dados representados no arquivo XML. Isto é possível porque um arquivo XML válido é sempre representável em uma estrutura de árvore, onde os nós corresponderão aos elementos dentro do XML;
- Finalmente, podemos usar a linguagem *Java* para acessar os dados representados em XML. Já existem várias API's para esta linguagem que fornecem fácil acesso aos elementos XML.

4.7. Vantagens da XML

A seguir temos algumas vantagens da linguagem XML na hora de escrever documentos estruturados.

4.7.1. Informação no cabeçalho

Um documento XML pode referenciar o seu DTD em seu cabeçalho. Isto significa que um arquivo XML já vem com seu “livro de regras” junto. Um programa processador de XML específico pode, desta forma, reconhecer se é capaz de processar um documento ou não. Enquanto isso, um browser HTML ignora qualquer elemento desconhecido. “(...) a filosofia do browser HTML é ignorar a DTD e também ignorar qualquer marcação que um processador em particular não entenda. Isto se traduz em força, mas a um alto preço de informação perdida.” [LIG99]

4.7.2. Classes de documentos

Usando a linguagem XML, podemos formar classes de documentos, isto é, documentos que compartilham um conjunto de *tag's* comum. Esta classe teria, então, um DTD genérico escrito para seus documentos e uma XSL também genérica. Tanto o DTD como a XSL poderiam ser estendidos para suportar outras *tag's* que viessem a ser adicionadas para processamento específico (somente definiríamos o núcleo da classe, mas nada impede que se use outras *tag's*). Visto desta forma, a HTML é uma classe de documentos XML e os browsers são softwares específicos escritos para leitura e processamento de HTML.

Se você possui uma classe de documentos em conformidade com uma única DTD, você tem de armar a completa aplicação XML somente uma vez - ao desenvolver as folhas de estilo, protocolos de links e possivelmente escrever software personalizado. Todo documento em conformidade com aquela DTD pode se beneficiar de sua aplicação associada. [IG99]

Uma outra vantagem no uso de classes de documentos é a facilidade de pesquisar informações. Como sabemos que todos os documentos possuem uma estrutura padrão, podemos criar um software que pesquise sobre esta estrutura. Tomando como exemplo uma classe para geração de documentos jurídicos, saberemos que todo documento da rede terá uma estrutura como: um elemento “processo” que contém um elemento “julgamento”, que contém um elemento “juiz”, que contém um elemento

“nome”. Assim, poderemos pesquisar em todos os servidores da rede, os processos julgados pelo juiz “José da Silva”. Poderíamos fazer uma consulta com a sintaxe: “processo[julgamento/juiz/nome=”José da Silva”]”. Isto nos retornaria somente os elementos “processo” julgados pelo juiz cujo nome é “José da Silva”. Todo o processamento é efetuado no servidor, o que diminui o tráfego na rede, pois só os dados que interessam são enviados.

4.7.3. Documentos longos

Um dos problemas da HTML é a escrita de documentos longos. Quando um usuário deseja uma informação que está em um documento HTML, precisa carregar o documento todo incondicionalmente. No XML podemos especificar um elemento a ser descarregado, transferindo o processamento para o servidor conseqüentemente diminuindo o tráfego na rede.

4.7.4. Foco na informação

Ao separar estrutura, dados e formatação, a XML tornou o ambiente da Internet uma grande base de dados. Afinal de contas, tudo que está na Internet (ou numa Intranet) é informação. Não apenas o que está contido no banco de dados sobre as vendas, mas também, por exemplo, o documento *MSWord* gerado pelo engenheiro contendo referencial técnico sobre a turbina de um avião, ou a ata da última reunião do sindicato, ou notícias sobre a bolsa da Ásia. Atualmente, utilizamos HTML para publicar estas informações. Desta forma, temos apenas uma apresentação “bonita” com informações interessantes. É ótimo para leitura de um humano. Mas e se precisarmos efetuar uma busca nestas informações?

Você já viu a atual geração de mecanismos de pesquisa da Web são ótimos de achar (‘sim! 15.000 itens!’), mas muito ruins em matéria de precisão (‘porque eu estaria interessado nisto?’). A XML já vem com um recurso de sensibilidade quanto ao conteúdo para a recuperação de textos. [LIG99]

Uma outra vantagem proveniente da separação de dados e formatação é o conceito de visões de documentos. Considere o caso de um documento contendo especificações técnicas sobre um determinado equipamento. Poderíamos ter pelo menos duas visões: uma voltada para o pessoal da montagem, exibindo passos e possíveis problemas na montagem e outra visão que exibisse modos de operação e possíveis problemas na operação do equipamento para o pessoal do suporte. A informação básica seria a mesma. A diferença seria apenas dentro dos ângulos de visão sobre os dados XML.

4.7.5. Estrutura que pode ser pesquisada

A XML torna a informação a ser disponibilizada na Internet, de uma certa forma, mais inteligente, pois agrega mais informação sobre a informação. Assim, o software que irá trabalhar com o documento poderá trabalhar com a informação de uma maneira mais útil do que simplesmente exibindo o documento formatado ao usuário.

Quando informação é guardada em uma aplicação XML específica, sua marcação pode refletir precisamente as semânticas daquela informação. Para começar, isso significa que pesquisar a Web pode se tornar um exercício muito mais preciso do que é atualmente. Também, o software na máquina local do cliente 'entenderá' a informação e será capaz de processá-la. [LIG99]

Por causa da informação num arquivo XML estar estruturada, o software que reconhece a XML poderá efetuar cálculos locais neste documento, sem a necessidade de voltar ao servidor para ter uma outra visão dos mesmos dados (como ocorre atualmente com a HTML). Por exemplo, considere o caso de uma grande empresa que mantém informações em XML sobre a folha de pagamento na Intranet. Um executivo da empresa pode acessar o documento e obter cálculos sobre o impacto da folha de pagamento no faturamento da empresa. Enquanto isso, um empregado pode acessar os mesmos dados para saber se o seu salário "será suficiente para pagar as contas".

4.8. Aplicações e Ferramentas

A seguir descrevemos algumas aplicações e ferramentas da linguagem XML.

4.8.1. Jumbo

A XML já vem sendo utilizada na WWW. Um exemplo deste uso é a CML (*Chemical Markup Language* – Linguagem de Marcação Química). A CML é uma linguagem definida para demarcar notações químicas em páginas WEB. Um DTD foi definido para validar um documento CML. Para exibição em browsers, geralmente utiliza-se *applet*'s. O Jumbo é um exemplo de *applet* desenvolvido para exibição gráfica de fórmulas químicas. Ver Fig. 17.

Jumbo (Peter Murray-Rust, diretor da Virtual School of Molecular Sciences, Nottingham University, England NG7 2RD, peter.murray-rust@nottingham.ac.uk) é um grupo de classes Java projetado para exibir aplicações XML. Este pacote foi originalmente criado para ser usado com uma variante da SGML chamada CML (*Chemical Markup Language* – Linguagem de Marcação Química). Descrita por seu autor como "HTML com mais química", a CML permite estruturas moleculares, análise, estruturas de cristais, e muitos outros aspectos da química molecular a ser modelada e exibida sem forçar o usuário a se envolver demasiadamente com codificação SGML. (...) Mais detalhes sobre o Jumbo podem ser encontrados no endereço <http://ala.vsms.nottingham.ac.uk/vsms/java/jumbo>. O pacote, inclusive um grupo completo de exemplos, está disponível gratuitamente, mas somente para uso pessoal. (Você pode adicionar novos exemplos, mas não pode redistribuir as classes sem a permissão do autor). Você pode descarregar do endereço <http://www.venus.co.uk/omf/cml>. [LIG99]

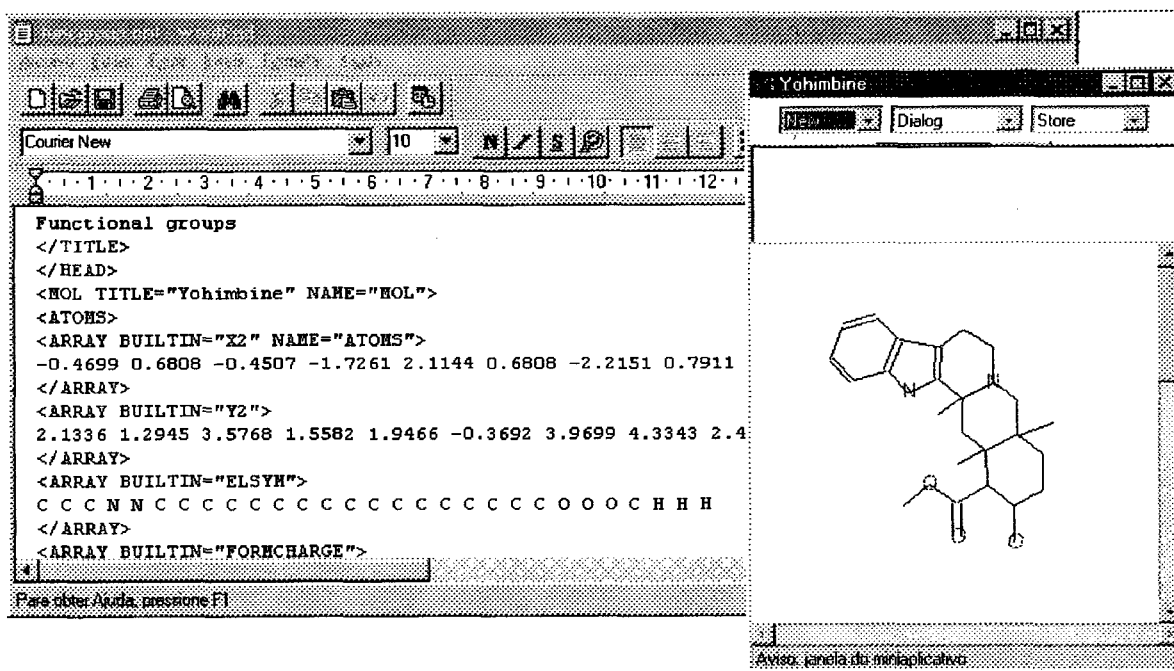


FIGURA 17: UMA MOLÉCULA REPRESENTADA NA APPLLET JUMBO. À ESQUERDA TEM-SE O CÓDIGO FONTE DO ARQUIVO.

4.8.2. MathML

A MathML é outro exemplo do uso da XML para notação científica. Destina-se à escrita matemática, sendo que já existem DTD's e *applet's* voltados ao uso desta tecnologia. Ela surgiu do problema de se representar expressões matemáticas em HTML.

Em abril de 1995, logo após a conferência em Darmstadt, Alemanha, (...) um grupo de partes interessadas se uniu para discutir o problema mais profundamente. Nos anos que se seguiram, o grupo informal tornou-se um grupo de trabalho formal do W3C, atraindo membros de fontes renomadas como a American Mathematical Society e Elsevier Science Publisher. Finalmente, em maio de 1997, o grupo publicou um esboço da especificação de uma aplicação XML chamada Mathematical Markup Language (Linguagem de Marcação Matemática) ou MathML. [LIG99]

4.8.3 Ferramentas de edição

Hoje já existem diversas ferramentas de edição que facilitam o trabalho de escrita de documentos XML e XSL. Exemplos destas ferramentas são:

- *eXML*: Um editor de documentos XML pequeno e versátil. Muito prático para criar e manter documentos XML, XSL (tendo em vista que o documento XSL também é um documento XML válido), e HTML válido. Seu modo de edição é baseado na árvore de elementos XML, sendo que também é possível efetuar mudanças manuais no código. Ver Fig. 18.

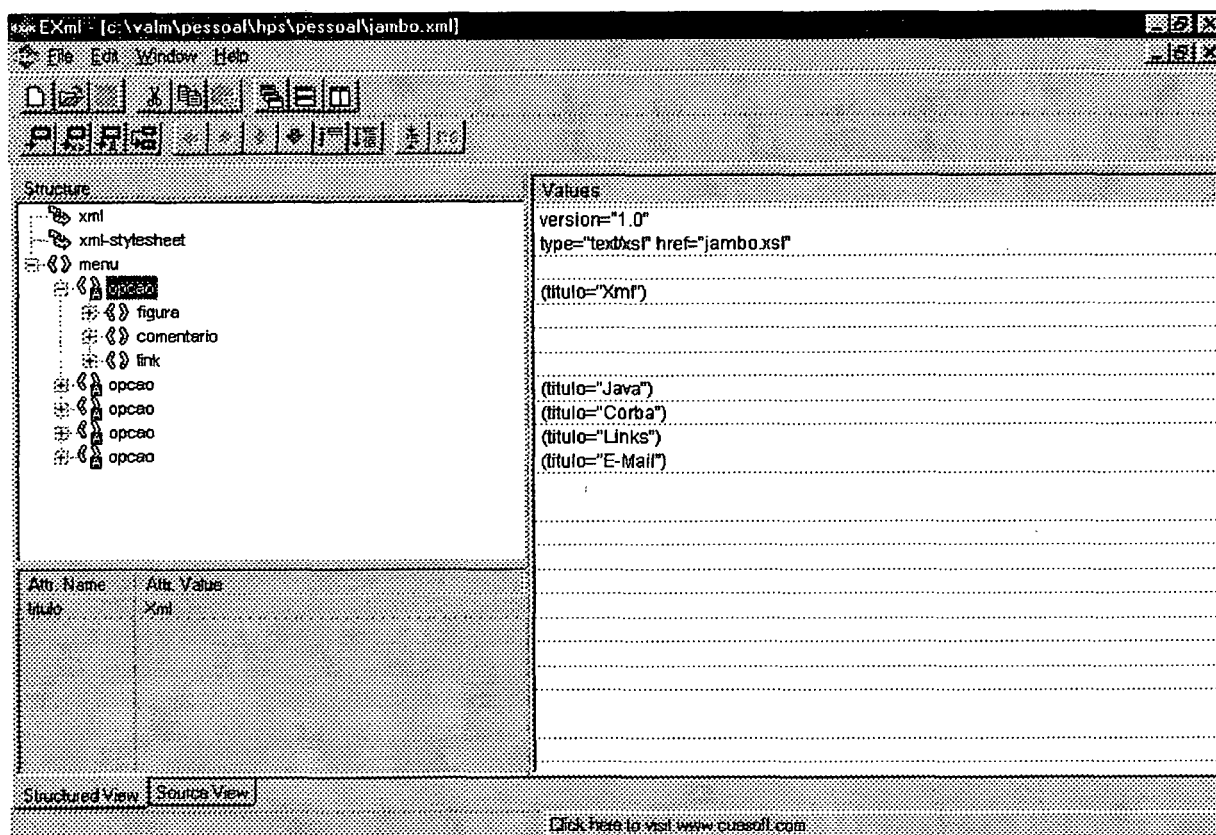


FIGURA 18: TELA DO EXML

- *XMLWriter*: o *XMLWriter* permite escrever documentos XML, XSL, HTML, DTD, CSS, além de documentos texto. A vantagem deste programa é que ele permite aplicar um arquivo XSL sobre um arquivo XML, gerando um HTML. Muito prático em diversas situações. Ver Fig. 19.

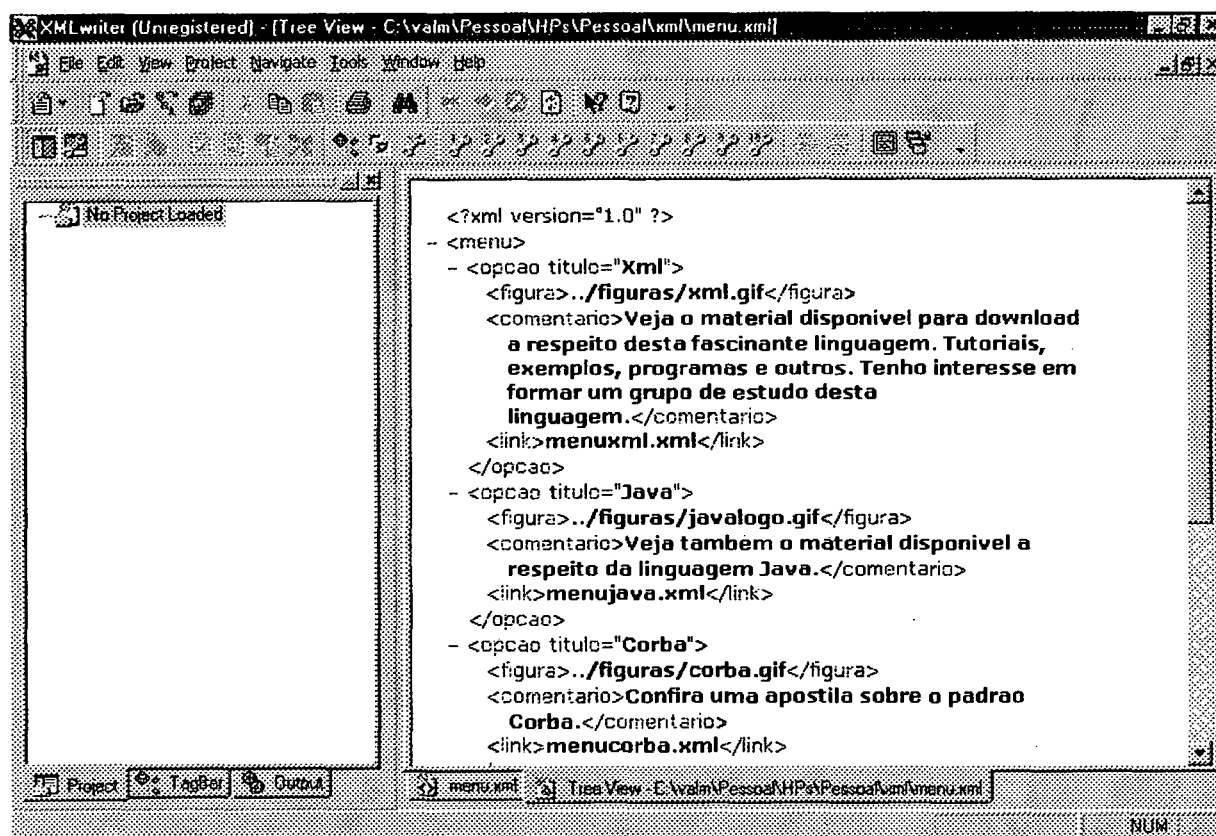


FIGURA 19: TELA DO XMLWRITER

- *IBM XSL Editor*: O *XSL Editor* também tem como finalidade escrever um arquivo XML juntando-o com o XSL, gerando um HTML válido. Ver Fig. 20.

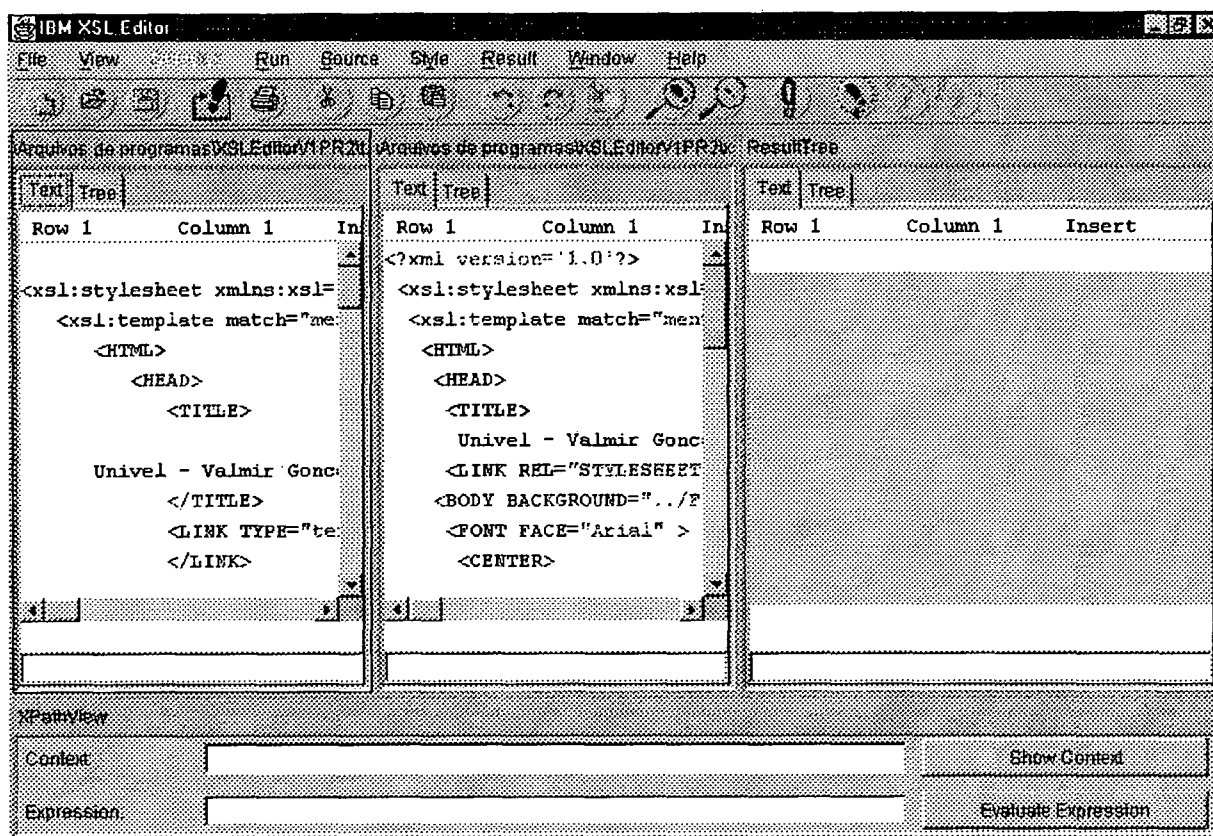


FIGURA 20: TELA DO IBM XSL EDITOR

Várias outras ferramentas já estão disponíveis atualmente no mercado. Algumas delas gratuitas.

No endereço <http://www.garshol.priv.no/download/XMLtools> podem ser encontradas várias ferramentas para XML.

4.9. Considerações Finais

A área de informática é marcada pelo surgimento repentino de novas tecnologias que se tornam rapidamente padrão de mercado. É uma área muito dinâmica em que as novidades surgem diariamente. A linguagem XML é uma dessas tecnologias. Surgiu recentemente e está rapidamente tomando seu lugar no mercado.

O surgimento da XML foi alavancado pelo tipo de informação que está sendo representada na Internet. Nos seus primórdios, a informação que estava disponível na

grande rede era puramente textual, sendo que a aplicação desta informação era mais científica. Sendo assim, não havia a necessidade de fazer a informação parecer atraente. Além disso, tudo que se desejava era transmitir textos com um pouco de formatação. Para isto, a linguagem HTML era ideal: fácil de aprender, compacta, fácil de interpretar e permitia formatar o texto de maneira satisfatória.

Porém, com o crescimento de uso pela comunidade em geral (incluindo a científica), verificou-se que a quantidade de informação a ser transmitida na WEB aumentou muito. Além disso, o conteúdo precisa ser atraente e fácil de visualizar. Sob este panorama, a linguagem XML desponta como uma alternativa inteligente, pois possui características capazes de atender a estas necessidades. Sua estrutura permite um processamento mais inteligente das informações a serem disponibilizadas na Internet.

Sob este ponto de vista, podemos afirmar que a linguagem XML é uma boa alternativa para a publicação de informações estruturadas.

5. CORBA COMO UMA TECNOLOGIA PARA ABSTRAÇÃO DOS CONCEITOS DE REDES ATIVAS NO NÍVEL DA APLICAÇÃO

5.1. Introdução ao padrão CORBA

A tecnologia de objetos distribuídos surge como alternativa para diferentes paradigmas de distribuição de processamento da informação. A abordagem cliente/servidor amplamente utilizada em aplicações distribuídas se mostra bastante poderosa no sentido de processar instruções compartilhadas, mas carece de mecanismos de encapsulamento de detalhes e a possibilidade das aplicações utilizarem memória em diferentes espaços de endereçamento. A utilização do conceito de objeto como unidade básica para a distribuição de processamento em substituição a unidade até então utilizada, o processo, se mostra eficiente para resolver os problemas acima mencionados.

A tecnologia CORBA (Common Object Request Broker Architecture) possibilita a utilização de objetos distribuídos em aplicações cliente/servidor. Essa tecnologia é independente de plataforma (sistemas operacionais e linguagens de programação) e da arquitetura do hardware utilizada, desta forma as diferentes entidades (cliente e servidor) que utilizam a tecnologia de objetos distribuídos podem ser desenvolvidas considerando os aspectos que melhor atendam aos seus requisitos [COR 95].

Através do padrão CORBA servidores de aplicações podem implementar objetos que podem ser usados remotamente por aplicações-cliente através de *interfaces* bem definidas. A especificação CORBA define que aplicações-cliente podem se comunicar com objetos que são implementados em servidores de aplicação. A comunicação é gerenciada pelo ORB (Object Request Broker).

5.2. Uma visão arquitetural do padrão CORBA

5.2.1. Object management group (OMG)

É uma organização internacional, com mais de 800 membros; incluindo vendedores de sistemas de informações, produtores de software e usuários. Fundada em 1989, a OMG promove estudos sobre a tecnologia orientada a objetos no processo de desenvolvimento de software. O carro chefe da organização é o estabelecimento de diretrizes para a indústria de software e a especificação do gerenciamento de objetos, objetivando oferecer um *framework* comum (CORBA) para o desenvolvimento de aplicações. Os objetivos primários são a reutilização, portabilidade e interoperabilidade de software baseado em objetos, em um ambiente heterogêneo e distribuído. A Fig. 21 mostra uma visão macro do padrão CORBA.

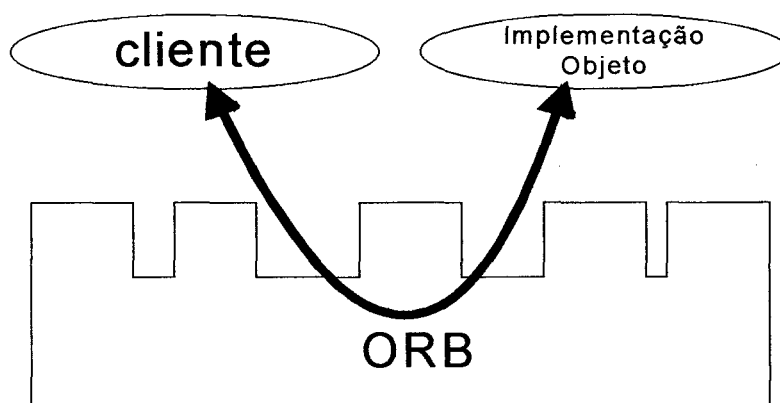


FIGURA 21: ARQUITETURA DE UM SISTEMA CORBA

5.2.2. A aplicação cliente

A aplicação-cliente emite pedidos para executar operações (métodos) sobre os objetos. O cliente possui uma referência para cada objeto alvo.

Quando o pedido do cliente é respondido, este pode então utilizar os resultados para realizar os seus objetivos. Caso contrário, se o pedido gerar um erro, este retornará ao cliente para que ele possa gerenciar as exceções.

Portanto, o papel do cliente é, simplesmente, requisitar serviços através da invocação de operações. A invocação pode ser feita estática ou dinamicamente, através de uma *interface* padronizada.

5.2.3. Invocação estática (STUB)

O *stub* no cliente é uma API de uma *interface* específica, por onde, o cliente envia um pedido à implementação do objeto (servidor) estaticamente. Quando se fala em invocação estática através do *stub*, fala-se em verificação de tipo e localização da implementação do objeto, tudo em tempo de compilação.

O *stub* obtém do cliente a referência ao objeto alvo (implementação do objeto) que então é passada ao ORB (descrito na seção xxx), o qual irá realizar a busca do objeto na rede.

Os *stubs* escondem os detalhes envolvidos na comunicação de rede necessária entre clientes e servidores. Através de sua utilização, a invocação de métodos de objetos remotos torna-se similar, em sintaxe, as chamadas de métodos de objetos locais ao processo cliente.

5.2.4. Invocação dinâmica

A API de invocação dinâmica permite a invocação e criação dos pedidos dos clientes em tempo de execução.

A *interface* de invocação dinâmica dá ao cliente a capacidade de, a qualquer momento, invocar qualquer operação sobre os objetos que ele possa acessar na rede. Incluindo-se os objetos que não se encontram mapeados no *stub*, objetos recentemente adicionados na rede ou descobertos através de alguns componentes pertencentes ao padrão CORBA.

O núcleo do CORBA, do qual falaremos adiante, é responsável por preparar os pedidos dinâmicos para que eles tenham exatamente a mesma forma dos pedidos estáticos antes que eles sejam enviados à implementação do objeto.

5.2.5. *Object request broker (ORB)*

Atua como um núcleo (barramento) comum através do qual um objeto-cliente pode enviar mensagens para o objeto-servidor.

O ORB é um padrão CORBA que oferece um grande conjunto de serviços de *middleware* para objetos distribuídos. O ORB permite que um objeto-cliente descubra uma implementação do objeto em tempo de execução.

No entanto, o ORB é muito mais do que um simples *middleware* cliente/servidor, pois inclui funcionalidades como RPCs (Remote Procedure Calls), MOM (Message-Oriented Middleware), procedimentos armazenados e serviços ponto-a-ponto.

Os métodos podem ser chamados de forma estática (definindo uma chamada de um objeto-cliente a um servidor em tempo de compilação) ou dinâmica (identificando um método de um objeto em tempo de execução).

A *interface* oferecida pelas *APIs* dos programas é de alto nível graças à utilização de uma linguagem neutra declarativa, denominada IDL, que será explanada posteriormente; permitindo combinar chamadas de procedimentos escritos em diferentes linguagens como C, C++ e Java.

Para que um objeto-cliente faça uma chamada a um servidor não é necessário que ele saiba onde está localizado este servidor. A chamada é passada ao ORB que se encarrega de localizar o servidor para atender ao cliente. Os servidores são vistos como objetos *proxy*, com total transparência local/remota.

O ORB utiliza informações no pedido do cliente para determinar a melhor implementação que satisfaça este pedido. Entre estas informações incluem-se a operação que o cliente está requisitando e o tipo do objeto da operação que será executada.

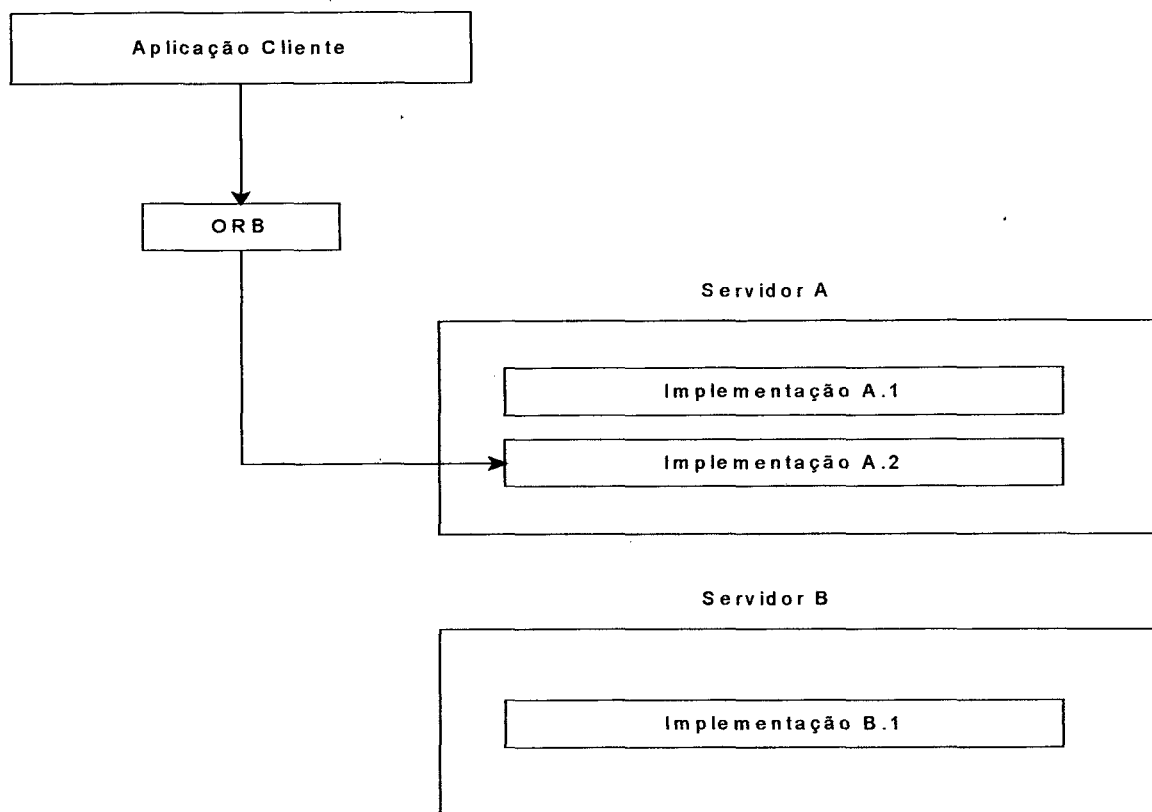


FIGURA 22: ORB SELECIONANDO UMA IMPLEMENTAÇÃO

Além de selecionar a implementação para efetuar o pedido - veja Fig. 22 -, o ORB também valida cada requisição e seus argumentos e pode fornecer mecanismos de autenticação ou informações de autorização. Por exemplo, o ORB analisa se o objeto especificado no pedido é válido para a operação que está sendo requisitada.

O próprio ORB funciona como um pseudo-objeto que possui operações definidas para desenvolvedores utilizarem em seus sistemas distribuídos. Como exemplo, temos a operação do lado do cliente **CORBA_ORB_create_list**, a qual cria uma lista de 'valores nomeados' para a invocação dinâmica de um pedido; ou do lado do servidor, a operação **CORBA_ORB_obj_to_string** para converter uma referência de objeto CORBA em uma *string*.

5.2.6. Contexto de objeto

O contexto de objeto é uma estrutura que contém informações sobre o cliente, ambiente ou circunstâncias de um pedido que não é passado como um argumento

formal para uma operação. O contexto de objeto armazena suas informações como uma lista de propriedades, a qual consiste de um identificador e um valor *string* associado. Por exemplo, *display* seria um identificador de um contexto de objeto que indica o dispositivo de exibição que o usuário final está utilizando, e *Xwindows* seria o valor *string* que está associado com o identificador *display*.

Ambos cliente e ORB podem ler e escrever no contexto de objeto, o qual é passado como um argumento implícito. O ORB pode usar as informações para selecionar uma implementação, influenciar o comportamento de um método, localizar um servidor ou optar por políticas de ativação do servidor.

A arquitetura CORBA especifica que as informações do contexto de objeto são opcionais, de forma que o ORB não depende das informações fornecidas.

O ORB procurará por um contexto de objeto somente se:

- Existir uma cláusula em OMG IDL definindo a operação;
- A aplicação-cliente usar a *interface* de invocação dinâmica, a qual requer o uso do contexto de objeto, ainda que este contexto seja nulo;

5.2.7. Repositório de interface

O repositório de *interface* contém descrições das *interfaces* das aplicações e os objetos de dados que existem na rede. A função primária do repositório de *interface* é fornecer informações para as requisições de invocações dinâmicas. O cliente usa o repositório de *interface* durante a invocação dinâmica para obter a assinatura da operação que deseja requisitar, incluindo qualquer tipo definido pelo usuário, o valor de retorno e os parâmetros da operação.

A aplicação-cliente acessa o repositório de *interface* como parte do processo de invocação dinâmica usando as operações que CORBA define para este propósito, como exemplo, **CORBA_InterfaceDef_describe_interface** que descreve o método a ser invocado dinamicamente.

Os desenvolvedores podem usar o repositório de *interface* para:

- Gerenciar a instalação e distribuição das definições da *interface* ao redor da rede;
- Fornecer interoperabilidade entre diferentes implementações de ORB;
- Fornecer verificação de tipo para a requisição da assinatura da operação;
- e
- Compilar *stubs* diretamente do repositório de *interface* ao invés dos arquivos IDLs.

5.2.8. *Interface definition language (IDL)*

O objetivo do padrão CORBA é oferecer uma arquitetura aberta, não proprietária, capaz de ser uma via única de comunicação disponível entre ambientes computacionais. Assim a arquitetura CORBA não estabelece uma linguagem padrão, podendo ter os seus elementos desenvolvidos em várias linguagens, dentre elas, C e C++.

As *interfaces* dos objetos são definidas em uma linguagem neutra, a *Interface Definition Language* - IDL. Assim os componentes escritos em IDL são portados através de várias linguagens, ferramentas, sistemas operacionais e arquiteturas de *hardware*.

Os componentes CORBA podem estar em qualquer ponto de uma rede, e para um objeto enviar uma mensagem para outro, o objeto-cliente não precisa saber onde o objeto-servidor se encontra. Não importa se o outro componente está na mesma máquina em um processo diferente ou numa máquina remota conectada via internet.

Um objeto-cliente também não precisa conhecer detalhes de implementação dos servidores, uma vez que a linguagem de programação e o compilador utilizado na implementação do servidor são transparentes. Além disso, um só componente pode funcionar ora como cliente, ora como servidor. Para isso, basta que os objetos tenham *interfaces* especificadas de acordo com a IDL.

A IDL é totalmente declarativa, onde se define *APIs* que podem ser implementadas em qualquer linguagem de programação que mapeie esse padrão. Hoje, já se pode fazê-lo em C, C++, Java e *Smalltalk*, por exemplo. Esse mapeamento nas

diversas linguagens de programação é especificado pelo padrão CORBA. A sintaxe da IDL é um subconjunto do C++, acrescido de definições que suportem a distribuição dos objetos [OMG 99].

O objetivo geral do padrão CORBA é dar a todos os seus componentes a mesma *interface* e oferecer as mesmas ferramentas a todos para que os objetos da rede possam ser utilizados indistintamente. Então, para um componente fazer uma chamada a outro componente, basta aquele, conhecer a *interface* IDL deste.

A arquitetura CORBA permite a existência de milhares de componentes na rede, para isso ela armazena todas as *interfaces* dos objetos disponíveis no repositório de *interface*.

Esse repositório contém as superclasses dos componentes existentes (*metadados*), permitindo que um objeto descubra outro de forma dinâmica, em tempo de execução, como já foi mencionado anteriormente.

O arquivo IDL pode gerar diversos arquivos quando compilado. Isto depende do fabricante do produto que suportará o padrão CORBA.

Entretanto, segue abaixo os principais arquivos que podem ser gerados a partir de uma especificação IDL, quando este for submetido a um compilador que efetua o mapeamento para a linguagem de programação. Ver Fig. 23.

- Arquivo de *Stub*: arquivo gerado com as definições na linguagem de programação;
- Arquivo Cabeçalho: arquivo gerado com definições de tipos para uma linguagem específica;
- Arquivo Skeleton: arquivo gerado que constrói o modelo de objetos de CORBA na linguagem de programação específica [ORF98].

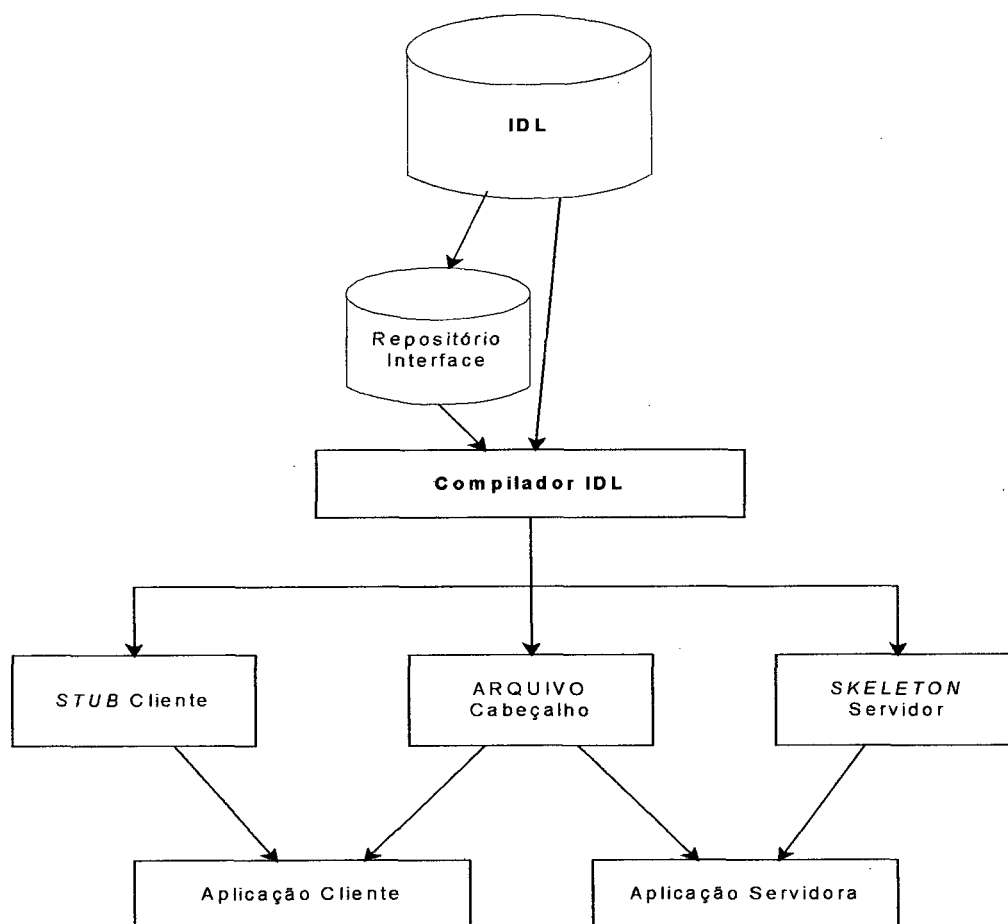


FIGURA 23: ARQUIVOS QUE PODEM SER GERADOS PELO COMPILADOR IDL

Abaixo um pequeno exemplo de código na linguagem IDL:

```

//Exemplo Produto
module T_Produto {
    typedef string entrada;
    typedef string saida;
    interface Produto {
        void Cadastrar (in entrada dados, in string desc,
                       in string quant, in string vlr);
        void Consultar (in string desc, out saida dados,
                       out string quant, out string vlr);
    }
}
  
```

```
void Excluir (in string desc);  
};  
};
```

QUADRO 9: LISTAGEM DE CÓDIGO NA LINGUAGEM IDL

Como se pode observar, o exposto no quadro 9 é bastante intuitivo no que se refere: às palavras reservadas e às definições impostas no exemplo.

Cabe agora ao compilador IDL gerar arquivos com o mapeamento adequado para a(s) linguagem(s) de programação. Observa-se que, para haver uma concordância com o padrão CORBA, o produto deve mapear a IDL para, pelo menos, uma linguagem de programação.

5.2.9. *Aplicação servidora*

A aplicação-servidora contém uma ou mais implementações de objetos que são as partes de um servidor que respondem aos pedidos dos clientes. Quando um cliente envia um pedido ao ORB, este seleciona uma implementação para satisfazer aquele pedido, de acordo com o que é solicitado pelo cliente.

Dentro da aplicação-servidora deve existir código para ativar a implementação do objeto, gerenciar os pedidos por serviços e desativar a implementação, se necessário; além de chamadas das operações CORBA. Ver a Fig. 24.

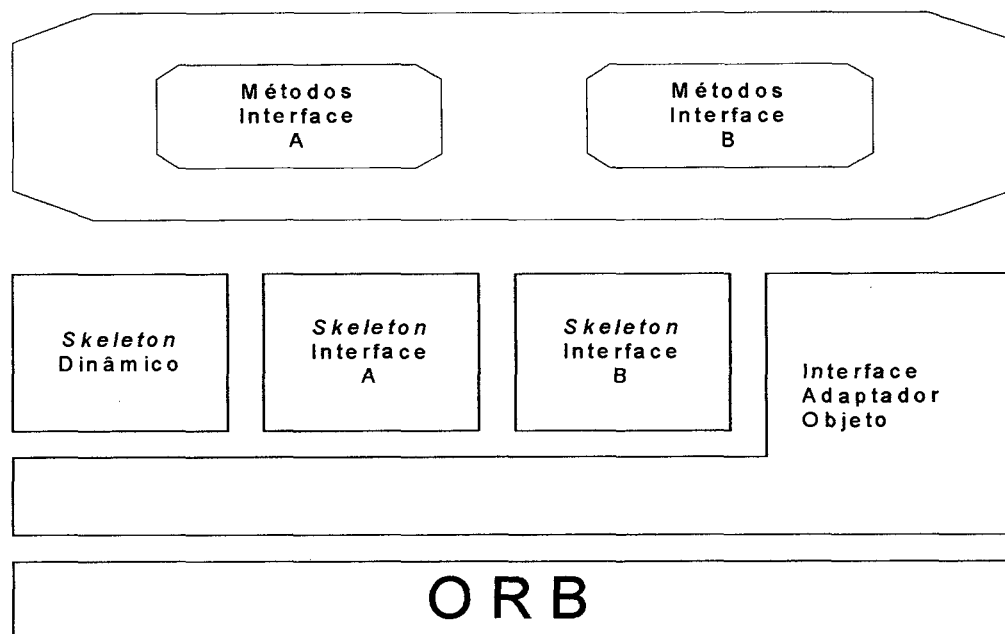


FIGURA 24: ESTRUTURA BÁSICA DE UMA APLICAÇÃO SERVIDORA

5.2.10. Adaptador de objeto

É um mecanismo primário de gerenciamento das referências de objetos e das implementações no servidor. O papel do adaptador de objeto é fornecer serviços do ORB para a implementação. Como os serviços de CORBA são variados, torna-se interessante haver vários adaptadores de objeto que tratam especificamente cada um desses serviços.

No entanto, o padrão CORBA requer, pelo menos, um adaptador de objeto nos produtos comerciais - o adaptador de objeto básico que fornece os serviços comuns.

Em sua essência, o adaptador de objeto deve fazer o seguinte:

- Fornecer uma ponte entre uma *interface* do objeto e a implementação deste no servidor;
- Gerar e interpretar as referências de objeto e mapear as referências para a implementação;
- Ativar e desativar objetos e implementações;
- Registrar implementações;

- Invocar implicitamente implementação de métodos.

Existem duas formas de implementar adaptadores de objetos: BOA (Basic Object Adapter) e o POA (Portable Object Adapter). [ORF98]

5.2.11. *Skeleton do servidor*

O *Skeleton* fornece a conexão entre o adaptador de objetos e os métodos que realizam cada operação sobre um objeto. *Skeletons* contém informações necessárias para mapear uma operação sobre um objeto à implementação apropriada. Eles são gerados a partir das definições OMG IDL e são específicos para *interface* e adaptador de objetos específicos.

Quando o ORB recebe um pedido, ele contata o adaptador de objetos através de um mecanismo que é privativo ao vendedor do produto ORB. Uma vez que o adaptador de objetos seleciona a implementação apropriada, ele notifica o servidor pelas chamadas às rotinas contidas no *skeleton* e este liga o adaptador de objetos à implementação apropriada.

Os *skeletons* atuam no processo servidor processando solicitações e executando métodos apropriados sobre referências de objetos. As referências para objetos e os métodos a serem disparados sobre elas são recebidos como parâmetros de clientes localizados remotamente.

Os *Stubs* (aplicação cliente) e os *skeletons* (aplicação servidora) fornecem o mecanismo pelo qual as aplicações CORBA escondem os detalhes envolvidos em chamadas a métodos de *interfaces*. Uma *interface* corresponde em semântica a uma classe abstrata, ou seja, apenas as assinaturas dos métodos são fornecidas (as *interfaces* geralmente residem em processos clientes, sendo suas respectivas implementações realizadas em processos servidores).

Assim para qualquer chamada ao método de uma *interface*, os argumentos são empilhados localmente e o método é disparado através de um ponteiro para a *interface* correspondente. Se a implementação do objeto não está no mesmo espaço de

endereçamento que a chamada ao método, o mecanismo de *stub* é ativado. O *stub* coloca os argumentos do método em um *buffer* e transmite estas informações via rede para o objeto remoto. Na aplicação servidora o *skeleton* recebe os argumentos transmitidos pelo cliente os empilha e chama a implementação do método desejado.

5.2.12. Repositório de Implementação

O repositório de implementação contém informações que permitem ao ORB localizar e ativar implementações de objetos. A arquitetura CORBA usa este repositório para associar a referência de objeto com implementações, assim como ele usa o repositório de *interface* para associar referências de objetos com suas *interfaces*. O repositório de implementação armazena as definições de implementações no pseudo-objeto **ImplementationDef**.

Boa parte das informações do repositório de implementação, pertence aos produtos que implementam o padrão e aos ambientes operacionais. Desenvolvedores e administradores podem utilizar as informações contidas no repositório de implementação para controlar políticas relacionadas à ativação e execução de implementações.

5.3. Interoperabilidade em CORBA

“A interoperabilidade ORB especifica uma aproximação compreensiva e flexível para suporte de redes de objetos que são distribuídos e gerenciados por múltiplos ORBs heterogêneos concordantes com o padrão CORBA” [OMG99].

A interoperabilidade é fornecida por vários elementos, dentre eles: o **protocolo geral Inter-ORB (GIOP)** e o **protocolo Internet Inter-ORB (IIOP)**.

“O GIOP especifica uma sintaxe padrão abstrata para transferência de informações (representação de dados em baixo-nível) e um conjunto de formatos de mensagens para comunicações entre ORBs” [OMG 99].

O protocolo GIOP é especialmente construído para interações entre ORBs e seu projeto fundamenta-se sobre o protocolo de transporte orientado à conexão.

O protocolo IIOP é uma especificação concreta da sintaxe abstrata GIOP usando conexões TCP/IP [BEN 98]. O IIOP é um protocolo para a Internet, que se relaciona com o GIOP da mesma forma que ocorre no mapeamento entre OMG IDL e as linguagens de programação.

A Fig. 25 exhibe o IIOP como uma realização concreta das definições GIOP sobre o protocolo TCP/IP, bem como as possíveis realizações para ambientes específicos.

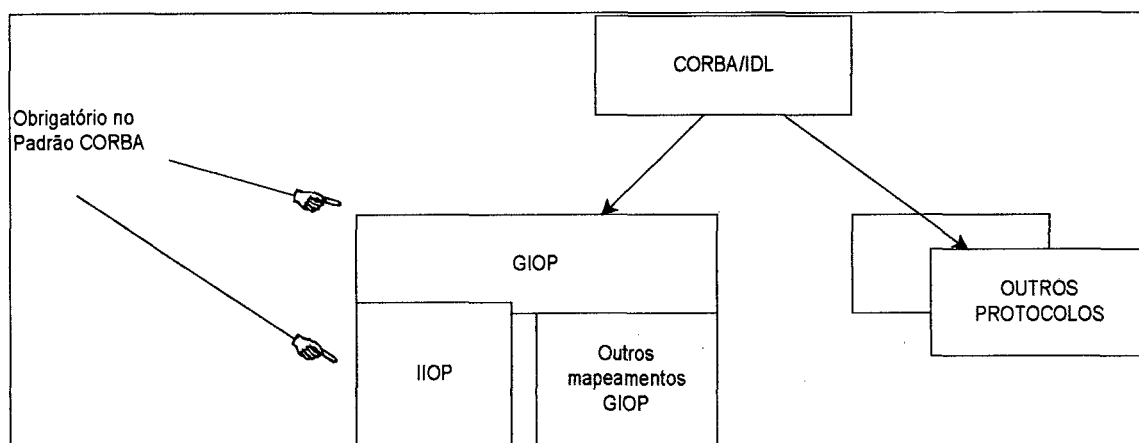


FIGURA 25: RELAÇÕES DOS PROTOCOLOS INTER-ORB

5.4. Serviços e facilidades do padrão CORBA

Os serviços CORBA são coleções de serviços ao nível de sistema empacotados como *interfaces* na linguagem IDL. Eles são um complemento da funcionalidade do ORB. Você utilizá-os para criar um componente, nomeá-lo e introduzi-lo em um ambiente. A OMG publicou os padrões para quinze serviços de objetos. Dentre eles temos:

- Ciclo de vida: operações de criação, cópia, movimentação e deleção de componentes.
- Persistência: armazenamento de componentes em uma variedade de *containers*.
- Nomeação: componentes que localizam outros componentes pelo nome.

- Evento: componentes registram ou desmarcam seu interesse em eventos específicos.
- Controle Concorrência: gerenciador de bloqueio em transações ou threads.
- Transação: coordenação do *commit* em duas fases entre componentes recuperáveis.

Com CORBA, os fornecedores de componentes podem desenvolver seus objetos sem qualquer preocupação com os serviços do sistema. Assim, o desenvolvedor cria apenas objetos que atendam as suas necessidades.

As facilidades de CORBA são coleções de *frameworks* em IDL que fornecem serviços para uso direto por objetos de aplicações. As duas categorias de facilidades comuns – horizontal e vertical – definem regras de compromisso que os componentes de aplicações necessitam cumprir para efetivamente colaborarem entre si, sendo que as horizontais têm como alvo proporcionar um *framework* para soluções ao nível da aplicação (agentes, *workflow*), enquanto as verticais visam soluções voltadas aos problemas do mundo real (financeiro, saúde).

As facilidades incluem agentes móveis, intercâmbio de dados, *workflow*, *firewalls* e *frameworks* para aplicações de diversas áreas, dentre elas: a financeira, a saúde e a telecomunicações.

5.5. Considerações finais do padrão CORBA

O padrão CORBA possui uma característica que lhe diferencia de qualquer outra tecnologia até então produzida em escala comercial, que é a independência de linguagens de programação, sistemas operacionais e arquiteturas de computadores.

CORBA engloba algumas técnicas já conhecidas em outras tecnologias como, por exemplo, as chamadas de procedimentos remotos e monitores; mas também inclui novos métodos que melhoram e facilitam a construção de aplicações distribuídas, além de contar com o parceiro infalível, os objetos.

6. DESENVOLVENDO O FRAMEWORK E A APLICAÇÃO

O presente trabalho desenvolve uma aplicação que abstrai conceitos de redes ativas, fundamentando-se no modelo arquitetural proposto por [FRY 98] e no trabalho de [MAR 99], porém utilizando o padrão CORBA como modelo para as chamadas de procedimento remoto.

Para isso, foi construído um *framework* ao nível de especificação que auxiliasse o desenvolvimento da aplicação.

Wirfs-Brock [WIA 91] define *framework* como “um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo *framework*”.

Um dos objetivos do *framework* é servir de base para diversas aplicações em um domínio de problema. Para isso, ele deve conter os conceitos envolvidos neste domínio. As interfaces criadas devem capturar tais conceitos de domínio de aplicações.

O *framework* proposto neste trabalho - veja quadro 10 - descreve na linguagem IDL/CORBA uma especificação para o desenvolvimento de aplicações que pesquisem dados em bases XML.

A linguagem IDL foi escolhida por apresentar diversas propriedades, tais como:

Declarativa – o especificador não necessita preocupar-se com os detalhes de implementação de cada aplicação;

Reutilizável – a especificação em IDL pode ser mapeada em diversas linguagens, logo, se desejado, poder-se-á reutilizar a especificação na construção de aplicações e de outros *frameworks* em linguagens específicas;

Orientada à especificação de objetos – IDL é uma linguagem que define interfaces e, quando submetida ao compilador IDL, fornece *frameworks*/aplicações automaticamente em linguagens específicas; minimizando, assim, o esforço de desenvolvimento destes artefatos de software.

6.1 Especificação do framework em IDL

```

module XMLQuery {
    struct node {
        string nome; // nome da tag em XML
        string valor; // conteúdo da tag XML
        node no_filho; // tag dentro da tag (subtag)
    };
    struct token {
        short Id_token; // identificador do token
        string valor; // conteúdo da pesquisa do usuário
    };
    struct lista_ligada_token {
        token no;
        lista_ligada_token ref_prox_no;
    };
    interface DocumentoXML {
        string ObtemColecaoXML(in string Pattern);
        node FiltraArvore(in node no, in lista_ligada_token consulta, in short pos);
        node GeraArvore(in string nome_arquivo);
    };
    interface Lista-Token {
        boolean eh_char_valido (in char c);
        lista_ligada_token gera_lista_token (in string consulta, in short posicao);
    }
};

```

QUADRO 10: LISTAGEM DA ESPECIFICAÇÃO DO FRAMEWORK

6.2 Considerações sobre a especificação

Este *framework* no nível de especificação oferece aspectos importantes para a construção de aplicações de pesquisa de dados XML.

Primeiramente, é definido um pacote chamado de *XMLQuery*, o qual agrega as *interfaces* e as estruturas de dados responsáveis por capturar os conceitos do domínio do problema.

A estrutura de dados **node** captura o conceito de um nó existente em documentos XML e serve para a construção da árvore de elementos (tag) em XML. A estrutura **token** é a base para transformação da consulta do usuário, no formato *pattern*, para uma representação que facilite a construção de aplicações que efetuam *parsers* nas consultas. Já a estrutura de dados **lista_ligada_token** visa representar uma lista ligada que represente a consulta do usuário; tornando a manipulação da mesma mais eficiente.

A *interface DocumentoXML* encapsula as operações necessárias para a implementação de aplicações de pesquisa de dados XML. Especificamente a operação **ObtemColecaoXML** responde às consultas vindas do usuário, retornando o conteúdo XML de forma textual (string). A operação **FiltraArvore** visa extrair uma sub-árvore da árvore que representa o arquivo XML. A operação **GeraArvore** cria uma representação em forma de árvore do arquivo XML, facilitando, assim, a manipulação dos elementos do arquivo.

A *interface Lista-Token* encapsula as operações para a geração de uma lista ligada da estrutura **token**, extremamente necessário para a manipulação mais adequada da consulta do usuário.

6.3 A aplicação

As tecnologias descritas até agora podem ser utilizadas em conjunto para a resolução de um problema em *intranets* de grandes empresas (e futuramente, a própria Internet) e, por conseguinte, o problema de se encontrar a informação que desejamos.

6.3.1 Justificativa

As ferramentas de busca atuais dentro de *intranets* e da Internet resumem-se, quando muito, a efetuar busca parcial dentro de documentos, como se procurássemos uma palavra dentro de um processador de texto. Assim sendo, a palavra “título” pode significar tanto um título de um livro quanto um título de cobrança ou mesmo um título ganho por um time de futebol. As experiências ocorridas com ferramentas de busca como o *Yahoo* (<http://www.yahoo.com>) ou com o *Cadê* (<http://www.cade.com.br>) mostram que este tipo de pesquisa não é satisfatório, freqüentemente retornando milhares de páginas como resultado de uma pesquisa simples. Além disso, estas ferramentas sempre retornam informações de que não precisamos, por não conseguirem distinguir o significado de palavras iguais, mas que não querem dizer a mesma coisa (como é o caso, por exemplo, da palavra “título”). Outro problema é o fato destas ferramentas consultarem um banco de dados próprio para efetuar estas pesquisas. Isto tem várias desvantagens. Uma delas é o fato do banco de dados precisar ser muito grande para gerar pesquisas eficientes.

Para resolver estes problemas, pode-se imaginar uma Intranet construída com páginas escritas em XML. Cada servidor de páginas teria arquivos XML e arquivos de estilo (XSL) para formatar os dados estruturados em XML a gosto do *Web designer*. O DTD que definiria a estrutura destes arquivos XML teria de ser pré-definido.

Em algum servidor existiria um aplicativo em *Java* que seria usado para efetuar pesquisas na rede (na Internet, este aplicativo poderia ser apresentado em forma de uma *applet* que substituiria ferramentas como o *Cadê*). Exemplos de pesquisas válidas seriam: “Encontre-me todos os documentos que tenham um elemento ‘título’ que contenham um elemento ‘valor’” ou “Encontre-me todos os subelementos do elemento ‘título’ da rede”.

Basicamente, a aplicação constitui-se de:

a - Servidor de aplicação de pesquisa e filtragem de dados XML

Este módulo é o núcleo da aplicação, ele encarregar-se-á de receber os pedidos dos clientes, extrair as informações em arquivos XML e enviar os resultados ao usuário.

O servidor terá a responsabilidade de inicializar o(s) método(s) que extrairão as informações que o cliente está requisitando.

Os métodos responsáveis pela consulta na base XML dividir-se-ão segundo as seguintes tarefas:

- *Análise sintática* sobre a base XML com a geração da árvore de objetos XML (árvore sintática).
- *Pesquisa* sobre a árvore de objetos com a extração dos objetos (subárvore) que o usuário requisitou (filtragem).
- *Geração* do documento XML que será enviado ao usuário.

b - A base dados XML

As páginas XML conterão uma estrutura pré-definida para todos os clientes, isto é, o mesmo arquivo XML. Outra característica importante é que elas possuem a mesma estrutura e esta é conhecida de antemão; como consequência disto, a pesquisa torna-se mais eficiente e menos complexa.

A visualização das páginas origem não precisa ser sempre a mesma, pois a formatação de uma página XML é dada por uma folha de estilos à parte que pode variar de máquina para máquina. O mesmo ocorre com a página resultante da pesquisa.

c – Aplicação no cliente

A aplicação no cliente fará parte da *interface* para o usuário emitir requisições ao servidor. Ela também terá a responsabilidade de gerar as requisições no padrão CORBA. A Fig. 26 ilustra essas aplicações.

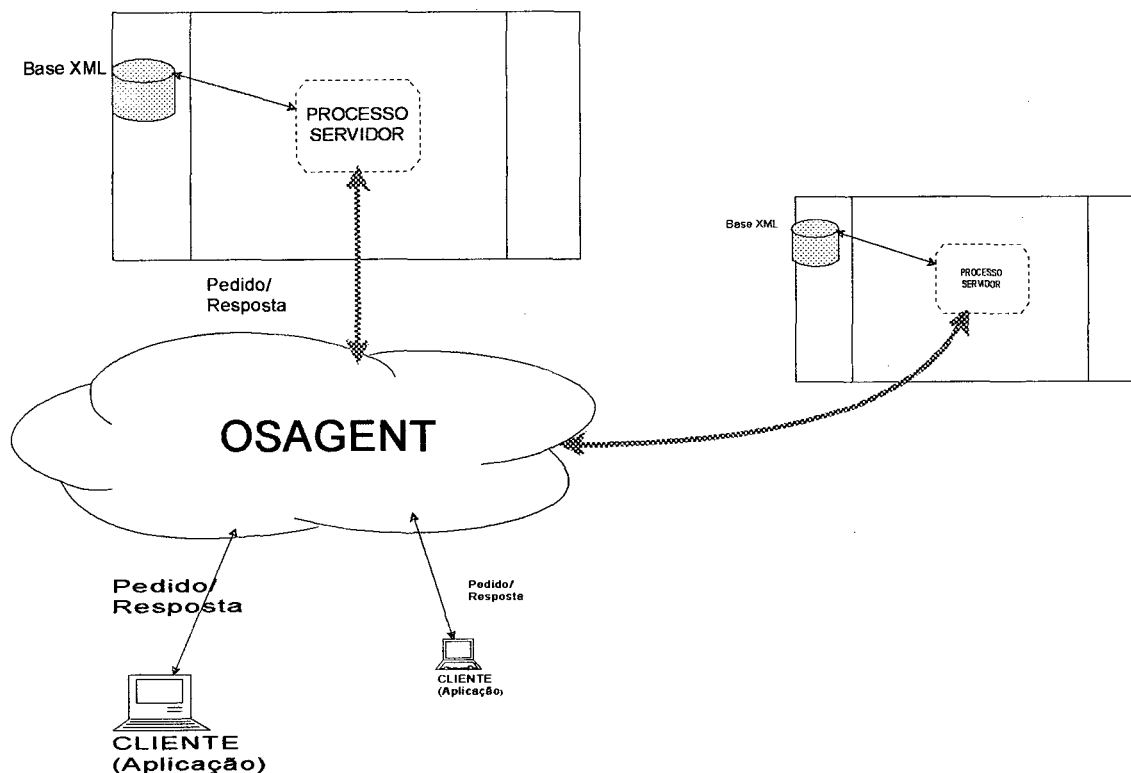


FIGURA 26: VISÃO DOS COMPONENTES DA APLICAÇÃO

Quando uma pesquisa é executada por algum usuário, ocorreriam os seguintes passos:

- Iº.) O aplicativo-cliente passaria a consulta ao ambiente CORBA (através do ORB) na rede;
- IIº.) O ORB localizaria o servidor que contivesse o objeto desejado para então executar seus métodos de busca, somente, nestes servidores que possuísem o elemento desejado. Isto porque, somente, estas máquinas possuiriam os objetos remotos publicados na rede;
- IIIº.) Os objetos retornariam os dados (resultados) ao ORB, e este, por sua vez, ao aplicativo solicitante;
- IVº.) O aplicativo montaria um documento com os resultados retornados. Este documento poderia ser acessado pelo usuário que fez a pesquisa, ou servir de base

de dados (servidora) para outros clientes, já que seria efetuado um *cache* na máquina cliente.

A Fig. 27 ilustra esses passos.

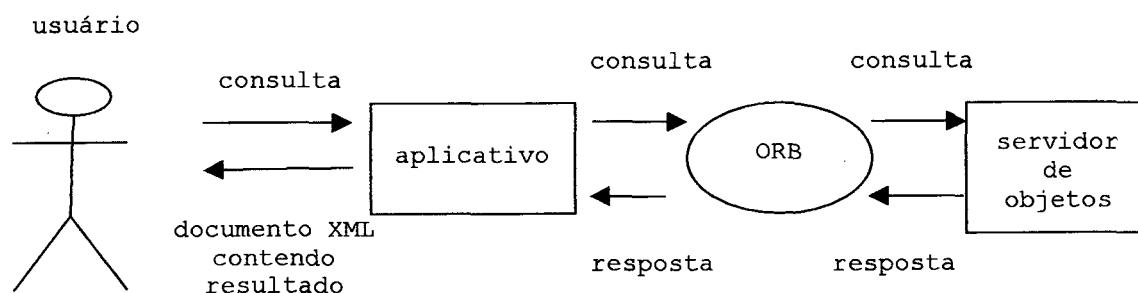


FIGURA 27: PASSOS QUE DESCREVEM COMO UMA CONSULTA GERA UM RESULTADO AO USUÁRIO

Para uma melhor compreensão do funcionamento do aplicativo desenvolvido, pode-se visualizar os diagramas UML no Anexo 1 confeccionados durante a modelagem do sistema, ou ainda, visualizar a especificação do *framework* em IDL/CORBA descrito anteriormente neste capítulo.

6.4. Materiais e método para construção da aplicação

O método utilizado segue o processo adotado no desenvolvimento desta dissertação. Em suma, inicialmente houve uma pesquisa bibliográfica que desse suporte ao entendimento dos conceitos da tecnologia XML, CORBA e Java, além dos conceitos de redes ativas.

Passado este primeiro momento, houve a preocupação de um estudo mais aprofundado das tecnologias CORBA e XML, as quais possuem ferramentas indispensáveis para o desenvolvimento de qualquer aplicação.

A partir daí, surgiu a necessidade de modelagem da aplicação em uma notação completa, clara e que desse suporte aos conceitos básicos das aplicações distribuídas. Neste caso, a linguagem UML[BOO00] surgiu para a realização da modelagem em alto nível de abstração.

No mesmo tempo que se definia os componentes na linguagem de modelagem, também, definia-se a estrutura da especificação do *framework* em IDL/CORBA.

Com o entendimento do problema em questão foi-se gerando versões da aplicação até chegar na versão atual que, por sua vez, satisfaz os objetivos principais deste trabalho mesmo não sendo ela completa e abrangente dentro do domínio do problema como um todo.

Após a sua conclusão, deu-se início as atividades de melhoramentos funcionais, no que se refere, ao desempenho durante a sua execução. A partir daí, houve alguns testes para medir o tempo de resposta de pedidos dos usuários, os quais estão descritos no subcapítulo - Avaliação da aplicação.

No sentido de avaliar melhor a aplicação proposta, foi desenvolvida outra aplicação que ao invés de consultar páginas XML, consulta páginas HTML. Com isso, teríamos algo para comparar algumas propriedades importantes em uma aplicação, principalmente, aplicações que integrem tecnologias como XML e CORBA. Esta aplicação é descrita sinteticamente no subcapítulo - A aplicação na prática - e no Anexo 2.

6.4.1. Ferramentas utilizadas

Para o desenvolvimento deste sistema foram utilizadas as seguintes ferramentas:

- *Java Development Kit 1.2.2;*
- *Inprise Visibroker 4.0;*
- *Java API for XML Parsing versão 1.0;*
- *Rational Rose 2000;*

6.4.2. Ambiente de desenvolvimento

Para o desenvolvimento da aplicação foi utilizado o seguinte ambiente:

- Estações com plataforma *Intel Pentium;*
- Sistema Operacional Windows 98/NT;
- Rede TCP/IP padrão *Ethernet 10 mbits.*

6.4.3. A aplicação na prática

A aplicação obtida permite disponibilizar vários arquivos XML para consulta numa rede. Os clientes irão consultar esta rede e obter arquivos XML que serão fragmentos dos arquivos disponibilizados na rede. Estes fragmentos serão a resposta para a consulta que foi efetuada.

Para que o aplicativo possa funcionar é necessário ativar o Agente Inteligente do *Visibroker* que irá providenciar a busca dos objetos que implementam os métodos que tratarão um pedido feito por um determinado cliente. No *Visibroker*, isto se faz através do comando *osagent*.

O passo seguinte seria inicializar o servidor. O servidor é o arquivo *Servidor.class* que exige como parâmetro o nome de um arquivo XML que será disponibilizado para consulta. Após obter a estrutura do arquivo, o servidor exibe na tela a mensagem "Aguardando pedidos".

O cliente pode ser invocado através do arquivo *Cliente.class* que exige dois parâmetros: a consulta a ser efetuada sobre um arquivo XML (numa linguagem baseada no formato *pattern*)[PAT00] e o nome de um arquivo onde será gravada a resposta obtida do servidor. Se este arquivo já existir ele será sobreposto.

A consulta é escrita no formato *pattern*, utilizada dentro de arquivos XSL. A linguagem de consulta *pattern* é semelhante ao formato utilizado no sistema operacional MS-DOS ou no MS-Windows para descrever diretórios. Consiste em nomear os nós desejados separados por uma barra (/). São exemplos de consultas:

biblioteca/livro/autor

Este exemplo irá retornar um arquivo XML contendo os elementos "autor" contidos em elementos "livro" contidos no elemento "biblioteca".

plantas/flores

Este segundo exemplo irá retornar um arquivo XML contendo os elementos "flores" que estiverem contidos em elementos "plantas".

O caractere suspenso (#) pode ser usado para designar todos os nós de um determinado nível. Um exemplo do uso do suspenso é dado abaixo:

biblioteca/livro/#

Este exemplo irá retornar um arquivo XML contendo todos os elementos que pertencem aos elementos "livro" que pertencem ao elemento "biblioteca".

No caso da linguagem *pattern* original, o caractere asterisco (*) exercia o papel de caractere curinga. Porém, no desenvolvimento da aplicação foi percebido que o compilador vbj (parte integrante do Visibroker) não reconhecia o caractere asterisco. Por este motivo, decidiu-se trocar o asterisco pelo suspenso.

6.4.4. *Teste da funcionalidade da aplicação em execução*

Para testar a funcionalidade do sistema considera-se a existência de um arquivo XML chamado *book-order.XML*, existente no mesmo diretório da aplicação, cuja listagem encontra-se no Anexo 3.

O arquivo *book-order.XML* pode ser visualizado no *Microsoft Internet Explorer*, conforme exibido na Fig. 28.

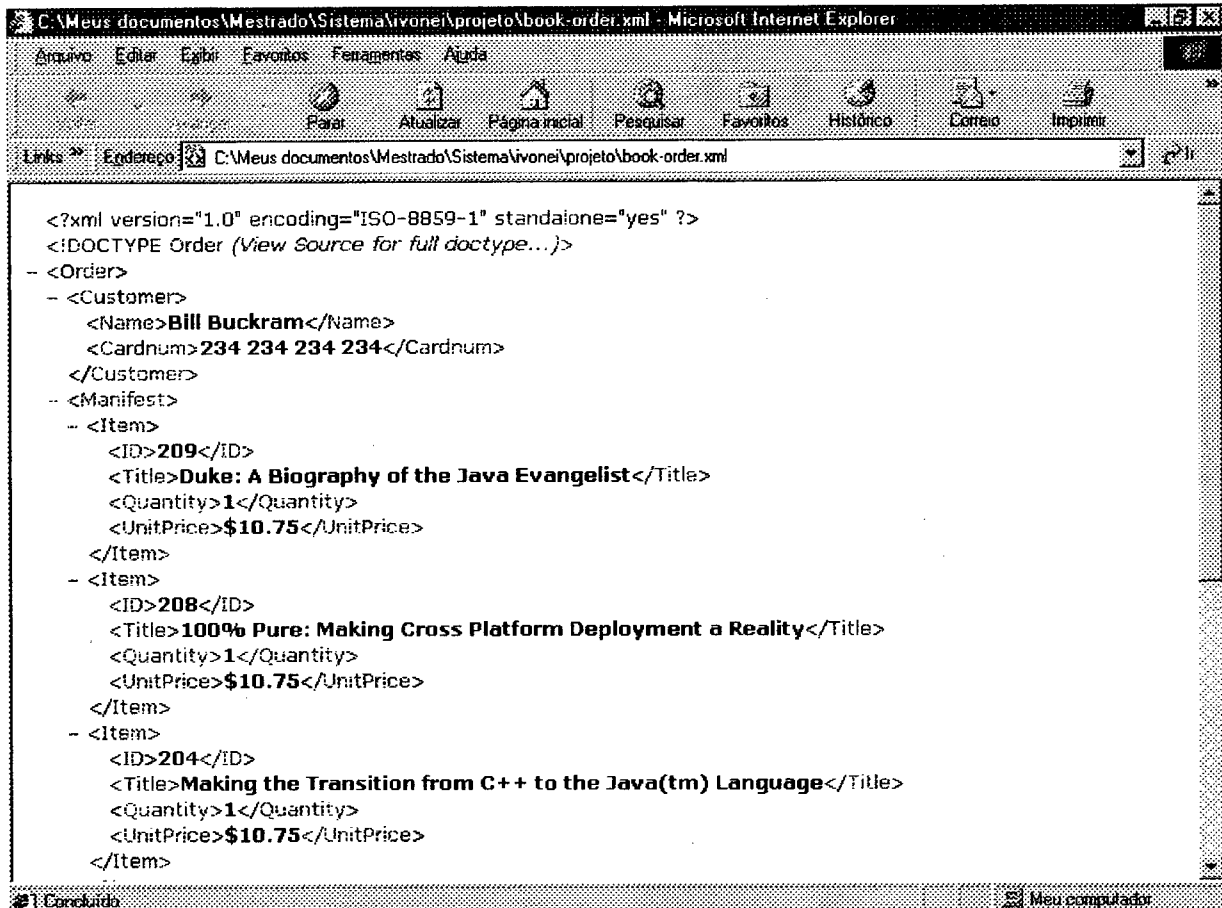


FIGURA 28: ARQUIVO BOOK-ORDER.XML VISUALIZADO ATRAVÉS DO MICROSOFT INTERNET EXPLORER 5.0

A aplicação desenvolvida é composta (principalmente, para o usuário) por dois arquivos: *Servidor.class* e *Cliente.class*. O primeiro disponibiliza o arquivo XML na rede local, e o segundo pode ser usado para efetuar a consulta a este arquivo.

Inicialmente, inicializa-se o agente inteligente do *Visibroker* em alguma máquina da rede local, através do comando *osagent*. Pode-se constatar a atividade do *osagent* através de um ícone que será adicionado à bandeja do sistema da máquina, conforme Fig. 29.

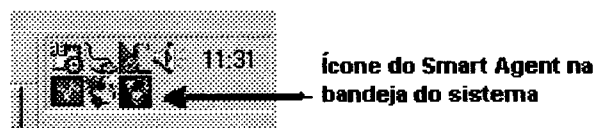


FIGURA 29: ÍCONE DO SMART AGENT

O próximo passo é inicializar o servidor de forma que este fique ativo e aguardando pedidos. Sobre o sistema Windows 98 podemos fazer isto através do comando *start*. O servidor exige um parâmetro que é o nome do arquivo XML, onde este deverá ser lido e, sobre o qual, as consultas serão feitas. Veja o comando e a tela que será exibida na Fig. 30.

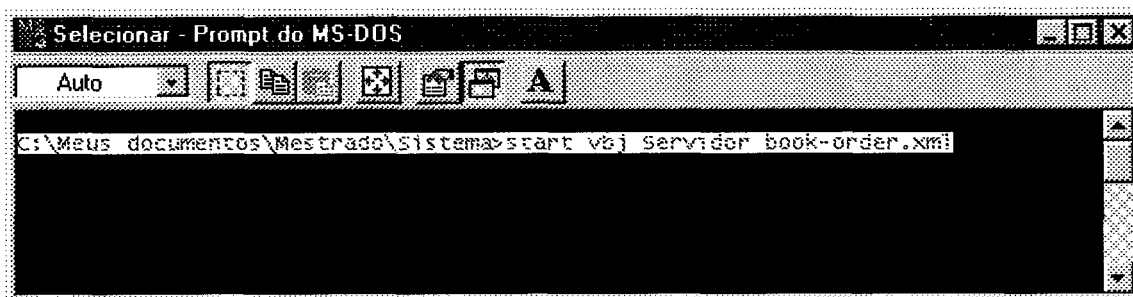


FIGURA 30: COMANDO QUE IRÁ INICIALIZAR O SERVIDOR DE OBJETOS

Após a execução deste comando, o sistema operacional Windows 98 abrirá uma nova janela na qual o Servidor será executado. Nesta janela, a estrutura do documento *book-order.XML* será exibida. Logo depois a mensagem “Aguardando pedidos” também será exibida, indicando que o servidor está pronto para aceitar requisições de clientes, conforme Fig. 31.

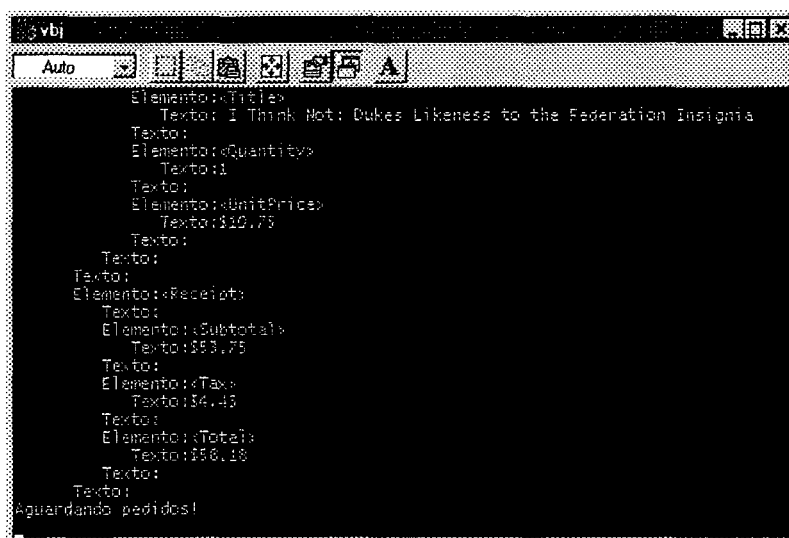


FIGURA 31: TELA DO SERVIDOR

O cliente pode ser acionado através do arquivo *Cliente.class*. Ele aceita dois parâmetros: a consulta e o arquivo onde o resultado será gravado. Veja um exemplo de consulta ao arquivo *book-order.xml* na Fig. 32.

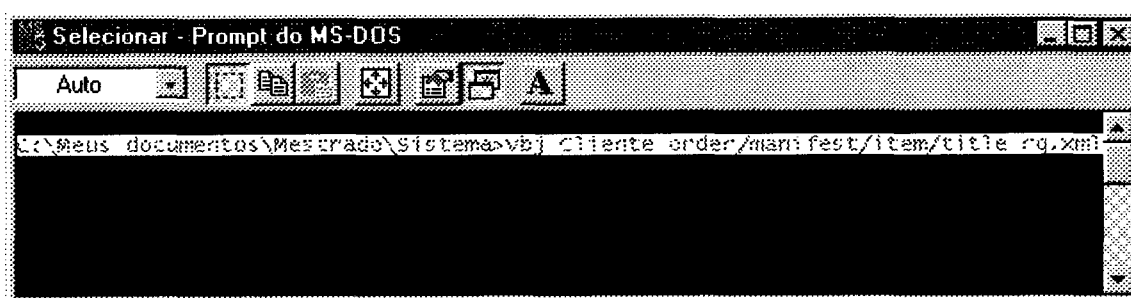


FIGURA 32: CLIENTE SENDO EXECUTADO

A resposta à requisição será gravada no arquivo *rq.xml*. Veja a saída do programa no arquivo *rq.xml*, visualizado através do *Microsoft Internet Explorer* na Fig. 33.

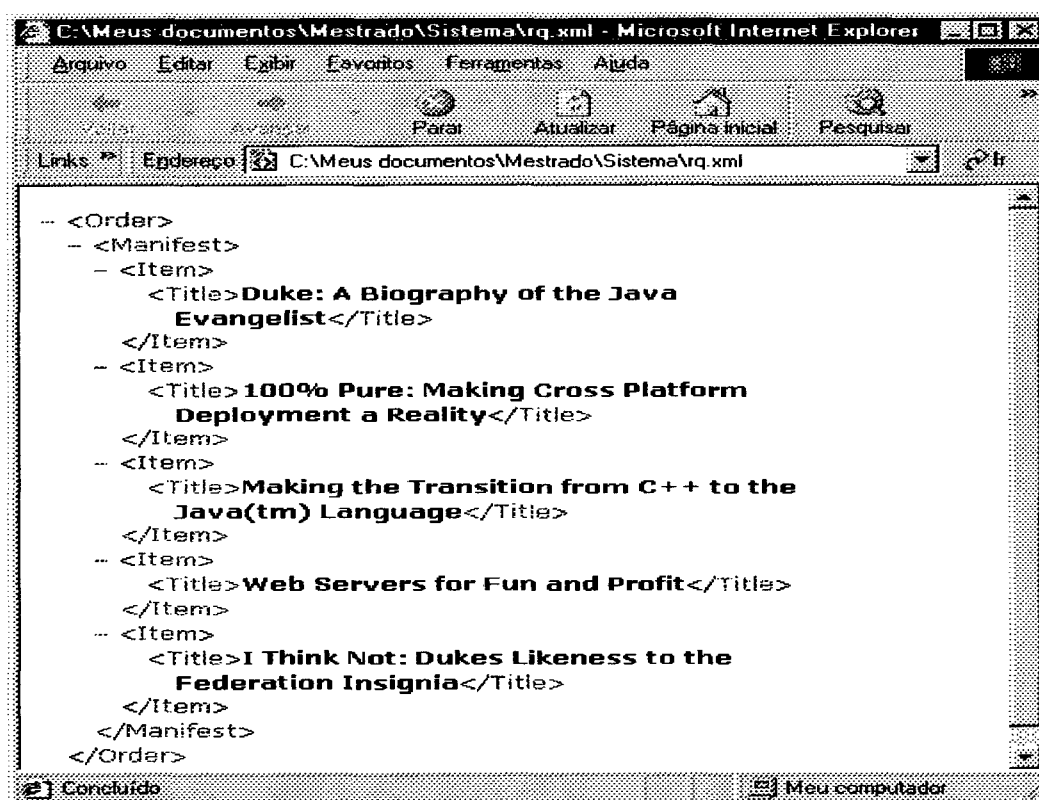


FIGURA 33: ARQUIVO QUE RESULTOU DA CONSULTA EFETUADA

Pode-se, também, efetuar uma consulta utilizando o caractere curinga (#). Veja na Fig. 34 um exemplo de consulta, onde fora consultado todos os elementos que pertencem ao elemento *Item*.

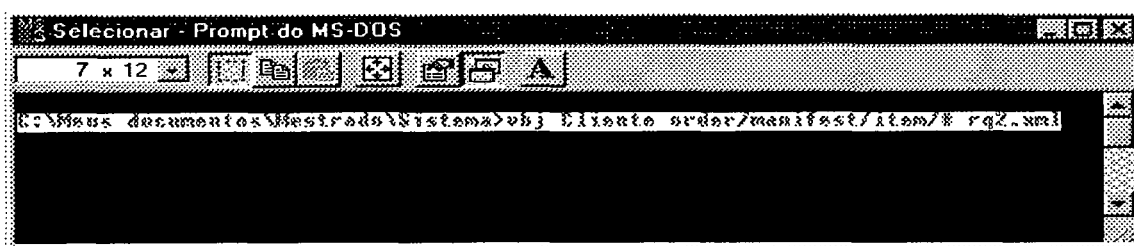


FIGURA 34: EXEMPLO DE UMA CONSULTA UTILIZANDO CARACTER CURINGA

O resultado será gravado num arquivo chamado *rq2.xml*. Pode-se visualizá-lo usando o *Microsoft Internet Explorer* conforme exibido na Fig. 35.

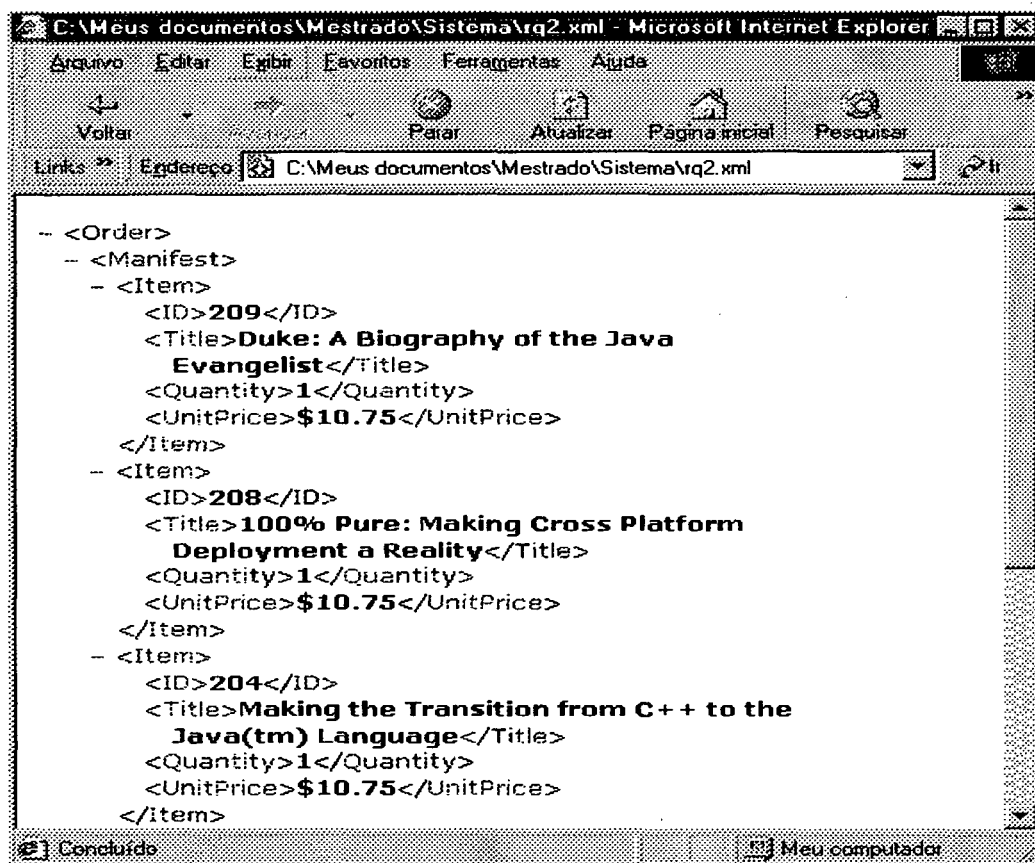


FIGURA 35: ARQUIVO RQ2.XML GERADO ATRAVÉS DE UMA CONSULTA AO SERVIDOR

6.4.5. Resultados: Medindo o desempenho

Na avaliação da aplicação que pesquisa/filtra dados XML deve-se considerar diversas variáveis para atingir conclusões realísticas sobre as suas funcionalidades. Dentre as variáveis, existe o tempo de processamento para abrir, pesquisar e filtrar os dados do arquivo em XML; este tempo está sendo computado juntamente com o tempo de resposta global da aplicação.

Para uma avaliação com uma maior completude e abrangência desenvolveu-se uma pequena aplicação de pesquisa de dados na linguagem HTML. Esta aplicação é semelhante à relatada neste trabalho com a linguagem XML, entretanto, ela diferencia-se pelo tipo de consulta (palavra-chave) e pela base de dados (dados em HTML).

Na aplicação que pesquisa textualmente uma palavra-chave em um arquivo HTML, existe o tempo que esta pesquisa leva para ser concluída, além da própria abertura do arquivo. Este tempo é computado no tempo de resposta global desta aplicação.

Ainda há uma outra variável que diz respeito ao tráfego de dados de outras aplicações, no caso, do próprio sistema operacional. Neste caso, procurou-se desabilitar todas as opções que porventura interferisse no desempenho das aplicações, ou mesmo, no tráfego entre a máquina cliente e servidora.

Então, para a mensuração é considerado o tráfego entre as duas aplicações (cliente/servidor em XML e HTML) e o processamento efetuado pelo servidor para buscar os dados solicitados pelo cliente, ou seja, é o tempo que se leva para chegar a resposta do servidor logo após a emissão do pedido do cliente.

O arquivo e os tempos adquiridos para a mensuração do desempenho estão descritos no Anexo 3 para o arquivo XML e no Anexo 4 para os tempos.

Na tabela 7 são descritos os itens considerados na medição do tempo de resposta da aplicação.

TABELA 7: TABELAMENTO DOS TEMPOS COM SUAS RESPECTIVAS CONFIGURAÇÕES

	Configuração A	Configuração B
Bytes XML	2.522	280.051
Bytes HTML	2.168	21.190
Média-Tempo(ms) XML	0.98	1.01
Média-Tempo(ms)HTML	0.99	1.19
Arquitetura	Pentium II 300 Mhz 128 MB RAM	Pentium II 300 Mhz 128 MB RAM
Sistema Operacional	Windows 98 se	Windows 98 se
Conectividade	Ethernet 10 Mb/s	Ethernet 10 Mb/s
Adaptador de Rede Cliente	Novell 2000	Novell 2000
Adaptador de Rede (Servidor)	Realtek RTL8029(AS)	Realtek RTL8029(AS)
Ambiente Execução/ Implementação	Visibroker 4.0 Jdk 1.2.2	Visibroker 4.0 Jdk 1.2.2

Descrição da tabela

Bytes (XML/HTML): representa a quantidade de bytes dos arqui/vos disponibilizados para consulta.

Média-Tempo(ms) XML: refere-se à média do tempo em milissegundos da aplicação implementada com XML.

Média-Tempo(ms) HTML: refere-se à média do tempo em milissegundos da aplicação implementada com HTML.

Arquitetura: representa a arquitetura do computador, essencialmente, processador e memória RAM.

Sistema Operacional: representa o sistema operacional adotado nas máquinas cliente e servidor.

Conectividade: refere-se à conectividade física da rede local.

Adaptador de Rede (Cliente/Servidor): refere-se ao adaptador de rede nas máquinas cliente e servidor

6.4.6. Análise dos resultados

Para esta avaliação não houve uma preocupação na igualdade exata dos dados nos arquivos XML e HTML, pois isto não interfere na mensuração de desempenho global da aplicação já que a performance é relativa às consultas emitidas tanto para os arquivos XML quanto para os arquivos HTML.

A pesquisa emitida para a aplicação em XML procurou extrair o menor número de dados para resposta à consulta emitida pelo cliente. Já na aplicação que consulta dados HTML isto, torna-se indiferente, já que o servidor irá retornar todo o conteúdo HTML disponível no arquivo.

Analisado os resultados nas tabelas no Anexo 4 percebe-se que a primeira medida chega a ser 30% superior em relação às demais medidas efetuadas logo na seqüência, ou seja, sem finalização da aplicação servidora. Isto é devido ao processamento de carga dos dados para a memória, tornando o tempo de execução nas aplicações servidoras mais demorado. A partir da segunda medição o tempo de resposta estabiliza-se, desde que a consulta seja a mesma.

Quanto às médias apresentadas na tabela 9, nota-se que a aplicação de pesquisa/filtragem de dados em XML é mais eficiente, principalmente, na comparação entre arquivos com maior quantidade de dados.

Logo, quanto maior o arquivo de dados, maior a tendência de se verificar uma diferença no tempo de resposta entre as aplicações. Isto é verificado já que todo o arquivo HTML é retornado; o que não acontece com arquivos em XML, onde os dados retornados dependem da estrutura e da consulta dos elementos XML. Desta forma, pode-se criar a estrutura de um arquivo em XML de tal forma que as consultas retornem apenas uma pequena quantidade de dados à aplicação cliente, independentemente, do tamanho do arquivo original. Esta é uma das principais vantagens de XML para a *WEB*.

Para a medição onde se verificou uma diferença considerável entre os tempos de resposta foi utilizado o arquivo `rich_iii.xml` com 280.051 bytes enquanto o arquivo `projeto.html` possuía 21.190 bytes, as listagens dos arquivos constam no Anexo 4. Já

nos arquivos book-order.xml (2522 bytes) e plantas.html (2168 bytes) houve uma pequena diferença nos tempos com vantagem para a aplicação que consulta dados em arquivos XML.

Verifica-se, portanto, que quanto maior o arquivo HTML maior será o seu tempo para ser enviado à aplicação cliente, e por conseguinte, maior será o tráfego na rede entre as máquinas servidora e cliente. Diferentemente, nos arquivos XML isto, não necessariamente se verifica, pois o tráfego utilizado dependerá da forma como o arquivo foi estruturado e da consulta emitida para este arquivo.

7. CONCLUSÃO FINAL

7.1. Considerações iniciais

A necessidade de novos serviços e aplicações para as redes de computadores estimulou um estudo detalhado de novos modelos de infraestrutura de redes de transmissão de dados.

Diante dessa realidade, este trabalho buscou alternativas para o desenvolvimento de aplicações distribuídas considerando o paradigma de Redes Ativas.

Observou-se que as Redes Ativas no Nível de Aplicação encaixa-se naturalmente nesta busca de soluções para o desenvolvimento de novos serviços.

Contudo, para que seja viável a implementação desse novo paradigma, necessitou-se de outras tecnologias para o desenvolvimento desses novos serviços.

A linguagem XML e o padrão CORBA apresentaram-se como alternativas extremamente viáveis na implementação de aplicações distribuídas para as redes de computadores.

Juntando-se essas tecnologias com a linguagem Java, teve-se um ambiente de desenvolvimento rico para a produção de aplicações distribuídas complexas.

Para a certificação destas suposições fora implementado uma aplicação que efetua pesquisa e filtragem de dados em arquivos XML e que mensura o tempo que se leva para a aplicação servidora do serviço responder à aplicação cliente. Com o intuito de haver uma referência para esta mensuração, implementou-se uma outra aplicação que consulta textualmente dados em arquivo HTML, cujo objetivo fora comparar o desempenho das duas aplicações e ratificar as vantagens levantadas pelos autores das tecnologias estudadas neste trabalho dissertativo.

7.2. Resultados alcançados

Os objetivos inicialmente traçados foram satisfeitos no nível teórico e no nível prático desta dissertação. Aspectos como: vantagens das tecnologias e abstração de conceitos de redes ativas são inerentes às próprias tecnologias estudadas (CORBA, Java e XML) o que facilitou o cumprimento dos objetivos.

Quanto ao aspecto prático da dissertação, nota-se também que os resultados das comparações entre os tempos de resposta atingiram os objetivos delineados no início deste trabalho.

Para isso, as pesquisas efetuadas em base XML resultaram em dados mais precisos, em menor quantidade de informações e, conseqüentemente, em diminuição do tráfego de dados na rede entre as máquinas cliente (solicitante do serviço) e a máquina servidora (prestadora do serviço de pesquisa/filtragem de dados).

Isto se verifica, principalmente, quando o arquivo de dados XML possui uma estruturação lógica entre os seus elementos facilitando a pesquisa e filtragem de dados lá contidos. Ou seja, é semelhante ao processo de compilação em uma linguagem de programação, o analisador léxico/sintático/semântico é aplicado a um conjunto de dados estruturados que segue regras específicas. Neste trabalho, o nosso “compilador” tem apenas as funções de pesquisa e filtragem de dados.

Ressalta-se também a importância do aspecto prático de modelagem da aplicação que consulta dados XML. O *framework* especificado na linguagem IDL/CORBA e o modelamento lógico na linguagem UML servem de base para futuras implementações que visem o melhoramento da aplicação até então implementada nesta dissertação.

7.3. Perspectivas

Nota-se que diversas ferramentas computacionais estão cada vez mais aderindo as tecnologias CORBA e XML. A nova versão do gerenciador de banco de dados

Oracle, inclui essas tecnologias como uma afirmação de que essas ferramentas possuem características desejáveis em diversas soluções para o mercado de *software*. A *Omg* é outra organização internacional que adota CORBA e UML como tecnologias essenciais para o desenvolvimento utilizando a orientação a objetos como base. Há um consórcio na WWW que estuda e padroniza a linguagem XML, inclusive a sua integração com tecnologias como CORBA, UML e Java.

7.4. Trabalhos futuros

De imediato, a ampliação de outros tipos de consultas aos elementos XML, semelhante às consultas executadas no padrão SQL. Exemplo: `/elemento_a/[elemento_b = 'x']/elemento_c`, cuja pesquisa objetiva obter o elemento_c, o qual encontra-se dentro de elemento_b que sejam iguais a x e, que por sua vez, estão contidos dentro do elemento_a.

- Especificação e implementação de novos serviços além da pesquisa e filtragem de dados. Exemplos: compressão e criptografia.
- Criação de Serviços/Facilidades CORBA do item anterior com posterior submissão a OMG.
- Integração da aplicação com os *browsers* existentes no mercado.

8. REFERÊNCIAS BIBLIOGRÁFICAS

- [ALE 00] Alexander et. al.. **Active network encapsulation protocol (ANEP)**, Draft of na Experimental RFC for the Active Networks Group. (Este memorando descreve o protocolo proposto para a padronização de tecnologias em Redes Ativas).
- [BEN 98] BENNETT, Geoff. **Internetworking com TCP/IP: tecnologia e infraestrutura**. Tradução Ernesto Veras, Rio de Janeiro: Infobook, 1998 2v.
- [BOO 00] BOOK, Grady; RUMBAUGH, James e JACOBSON, Ivar . **Uml, guia do usuário**. Tradução: Fábio Freitas da Silva – Rio de Janeiro: Campus, 2000.
- [COR 95] **The Common Object Request Broker: Architecture and Specification**, Revision 2.0 Julho 1995, <http://www.omg.org/CORBA/corbiop.htm>.
- [COR 00] **OMG/ISSO Standards**. www.CORBA.org/standarts.htm. 20 de dezembro de 2000.
- [DAR 00] **Defense Advanced Research Project Agency**, <http://www.darpa.mil>.
- [DEL 99] DELGROSSI, Luca; FATTA, Giuseppe; FERRARI, Domenico; RE, Giuseppe Lo. **Interference and Communications among Active Network Applications**. Italia. Notas de Leitura em Ciência da

Computação. Berlin, Alemanha: Springer. First International Working Conference, IWAN'99. Junho/Julho de 1999.

- [ECK 00] ECKEL, Bruce. **Thinking in Java**. <http://www.Bruce.Eckel.com>.
- [FRY 98] FRY, Michael e GHOSH, Atanu. **Application level active networking**. Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH'98), June 1998. <http://dmir.socs.uts.edu.au/projects/alan/prog.HTML>. 09 de dezembro de 1999.
- [JAV 00] Site de JAVA. **The Source for Java Technology**. Disponível na Internet. <http://www.javasoft.com> . 16 de fevereiro de 2000.
- [KUL 97] KULKARNI, A. B. et. al., **Implementation of a prototype active network**. Department of Electrical Engineering and Computer Science, Universty of Kansas, Lawrence KS 66045.
- [LIG 99] LIGHT, Richard. **Iniciando em XML**. Tadução Neilande de Moraes; revisão Roberto Gabriel Labrada; São Paulo: Makron Books, 1999.
- [LI- 98] LI-WEI, J.L. et. al. **Active reliable multicast**. IEEE INFOCOM'98 San Francisco, USA 1998.
- [MAR 99] MARSHALL, Ian et. al. **Active information networks and XML**. First International Woking Conference, IWAN'99. Berlin, Alemanha: Springer. Junho/Julho de 1999.
- [McG 99] McGRATH, Sean. **XML – Aplicações práticas**. Tradução Vitor H. P. Alves, Rio de Janeiro: Campus, 1999.
- [NEW 97] NEWMAN, Alexander. **Usando Java**. Rio de Janeiro: Campus, 1997, p. 19, 20, 21, 22.

- [OLI 97] OLIVA, Alexandre. **Programando em Java**. II Simpósio de Linguagem de Programação (SBLP'97). Instituto de Computação. Campinas: Unicamp, 1997.
- [ORF 98] ORFALI, Robert and HARKEY, Dan. **Client/server programming with Java and CORBA**. 2nd ed., Wiley Computer Publishing, 1998.
- [PAT00] **XML Query Language (XQL)**.
<http://www.w3.org/TandS/QL/QL98/pp/xql.html#XML> Patterns. 10 de dezembro de 2000.
- [RMI00] Sun Microsystems, Inc. **Java remote method invocation – distributed computing for java**.
<http://java.sun.com/marketing/collateral/javarmi.html>. 08 de dezembro de 2000.
- [SCO 98] SCOTT D. Alexander, HICKS W. Michael, KAKKAR Pankaj. **The SwitchWare Active Network Implementation**. Universidade da Pensilvania, Filadelfia, Setembro de 1998. Disponível na Internet.
<http://www.cis.upenn.edu/~switchware>. 08 de janeiro de 2000.
- [TEN 96] TENNENHOUSE, David L.; WETHERALL, David J.. **Towards an Active Network Architecture**. Laboratório de Ciência da Computação, Instituto de Tecnologia de Massachusetts – MIT. 1996. Disponível na Internet.
<http://www.sce.carleton.ca/netmanage/activeNetworks/mmcn96.HTML>.
10 de dezembro de 1999.
- [TEN 97] TENNENHOUSE, David; et. al. **A survey of active network research**. **IEEE Communications Magazine**, 35(1): 80-86, 1997.

- [VAN 00] VANASSI, Daniela. **Integração de redes ativas com agentes móveis.** Trabalho Individual, CTC – UFSC, 2000.
- [W3C 00] **HyperText Markup Language.** <http://www.w3.org/MarkUp/>. 11 de novembro de 2000.
- [WET 96] WETHERALL, David J. e TENNENHOUSE, David L. **The active IP option,** Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland, Sept. 1996.
- [WIA 91] WIRFS-BROCK, A. et al. **Designing reusable designs: experiences designing object-oriented frameworks.** In: Object-Oriented Programming Systems, Languages and Applications Conference; European Conference on Object-Oriented Programming, 1991, Ottawa. Addendum to the proceedings. Ottawa: [s.n.], 1991.

ANEXO 1: MODELAGEM DA APLICAÇÃO DE CONSULTA DE DADOS XML NA LINGUAGEM UML

Aqui estão todos os diagramas construídos para modelar a aplicação de pesquisa/filtragem de dados. Estes diagramas também deram suporte a construção do *framework* na linguagem IDL/CORBA.

O diagrama de use-case, Fig. 36, visa mostrar os requisitos funcionais do sistema.

DIAGRAMA DE USE-CASE

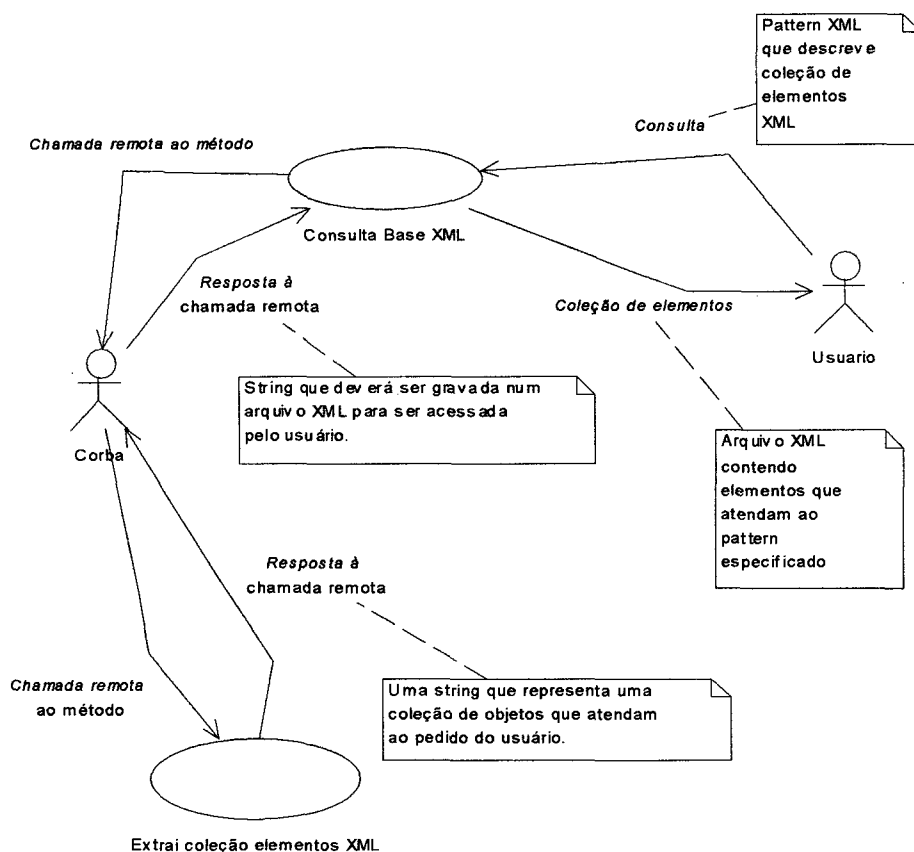


FIGURA 36: DIAGRAMA DE USE-CASE DA APLICAÇÃO

O diagrama de classes, Fig. 37, visa modelar a estrutura estática do sistema, especificando as classes que abstraem a estrutura conceitual do problema.

DIAGRAMA DE CLASSES

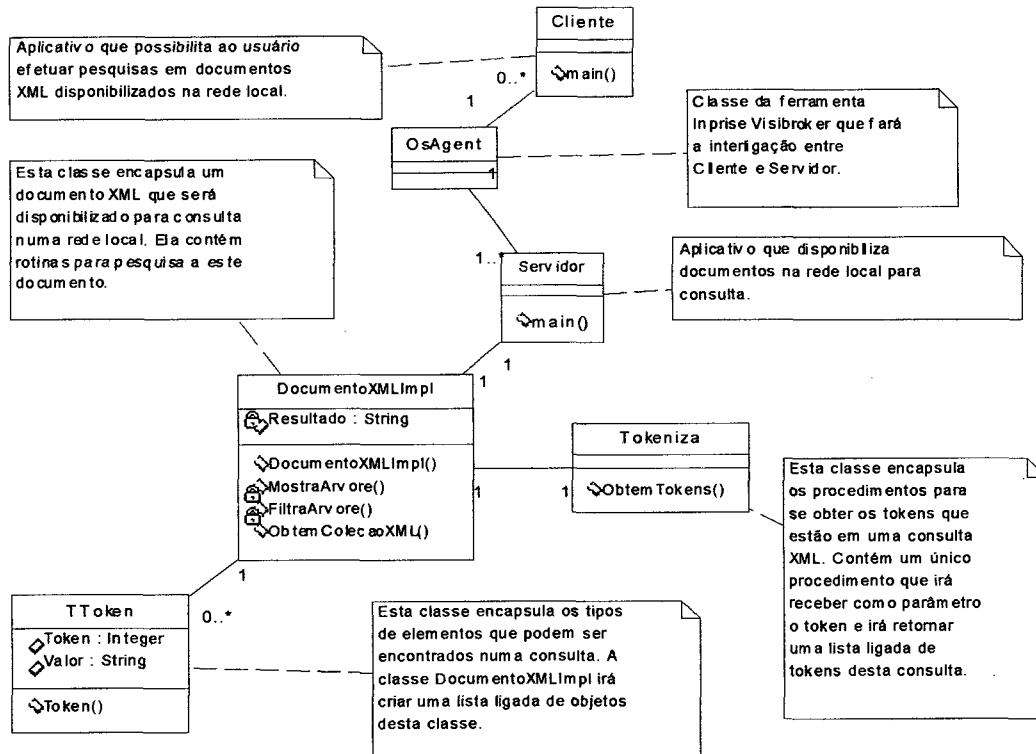


FIGURA 37: DIAGRAMA DAS CLASSES DA APLICAÇÃO

O diagrama de seqüência, Fig. 38, visa modelar a seqüência de interação entre os objetos de parte ou de todo o sistema. Neste caso, ocorre a seqüência, em um alto nível de abstração, de execução do sistema, desde a emissão do pedido pelo cliente até a chegada da resposta via ORB-OsAgent.

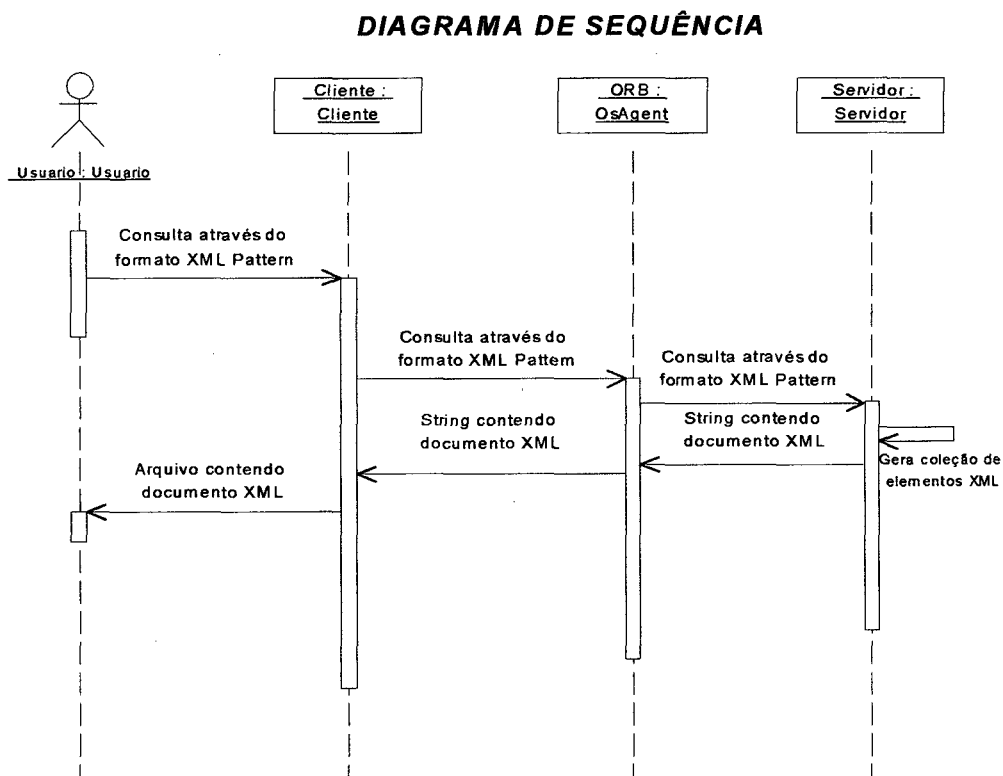


FIGURA 38: DIAGRAMA DE SEQUÊNCIA DA APLICAÇÃO

Os diagrama de componente e execução, Fig. 39 e 40 - respectivamente, exhibe o sistema em si, no nível computacional. Ou seja, os arquivos correspondentes gerados pelas ferramentas de desenvolvimento e também a topologia computacional de execução do sistema.

DIAGRAMA DE COMPONENTES

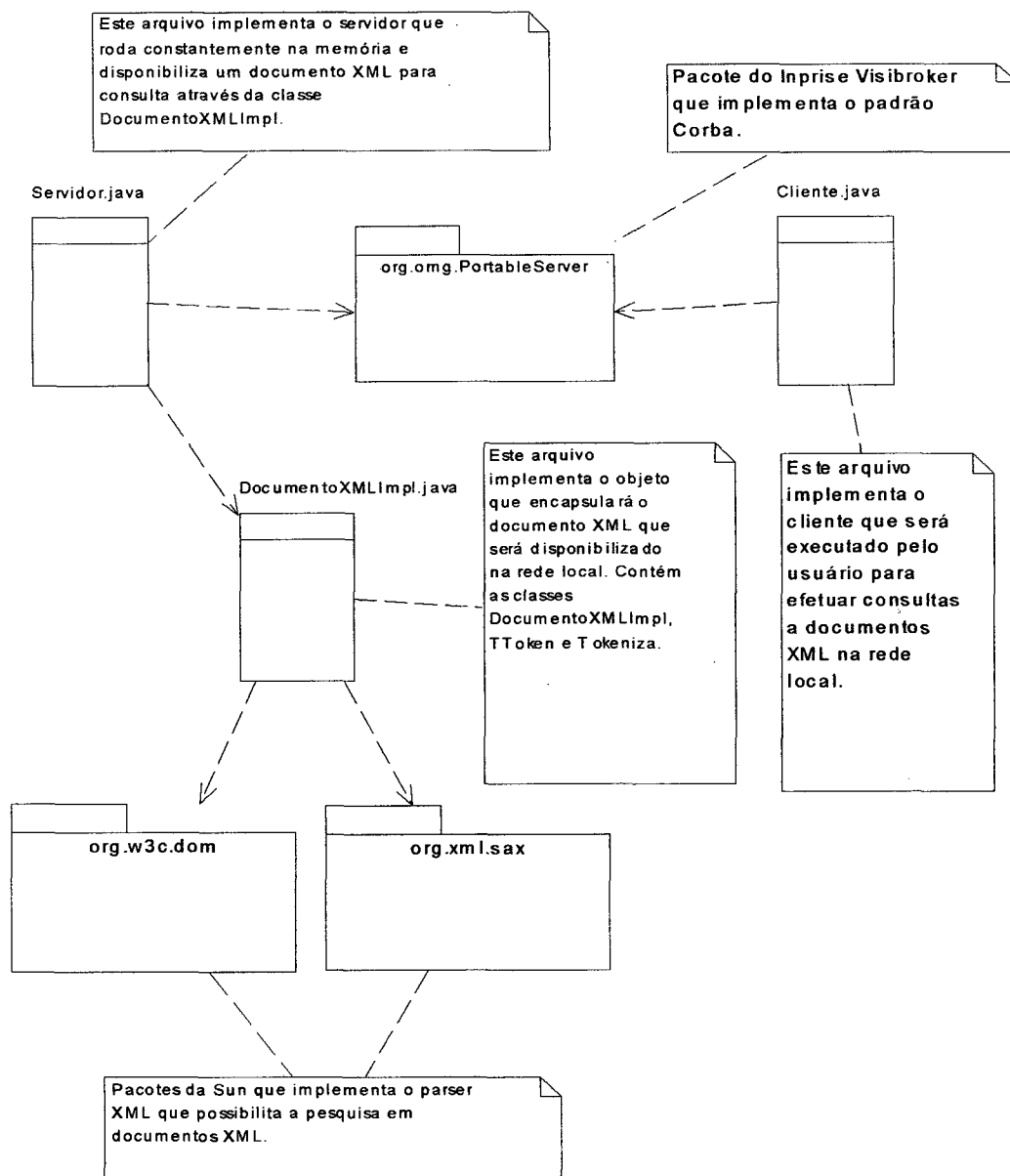


FIGURA 39: DIAGRAMA DOS COMPONENTES FÍSICOS DA APLICAÇÃO

DIAGRAMA DE EXECUÇÃO

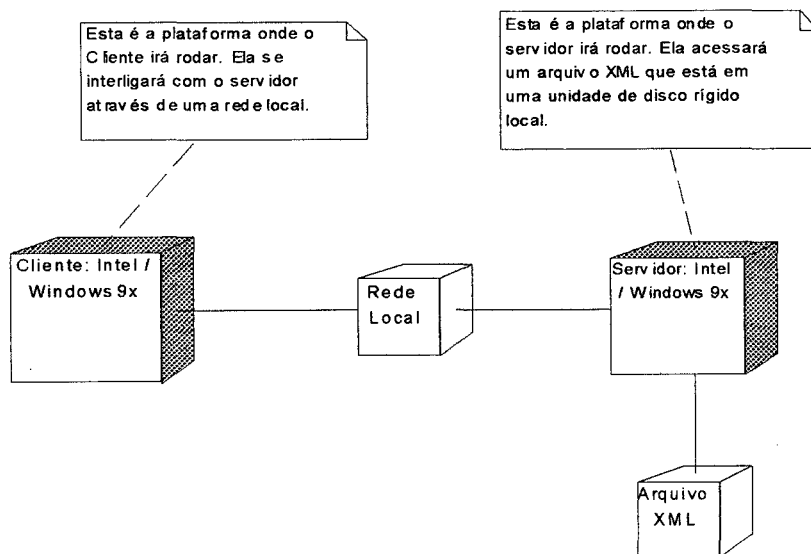


FIGURA 40: DIAGRAMA DE EXECUÇÃO DA APLICAÇÃO

ANEXO 2: CODIFICAÇÃO NA LINGUAGEM JAVA DA APLICAÇÃO DE CONSULTA DE DADOS XML

Aqui, serão apresentadas as classes implementadas no sistema. Quadros 11, 12, e 13. Bibliotecas utilizadas a parte, constam nas referências bibliográficas para maiores estudos. Ver quadro

A APLICAÇÃO CLIENTE

```
// Arquivo : Cliente.java
// Autor   : Ivonei Freitas da Silva
// Local/Data: Cascavel, 2000
// Classes  : Cliente

import org.omg.PortableServer.*;
import java.io.*;

// Classe: Cliente
// Variáveis públicas: nenhuma
// Rotinas públicas : void main(String[])
// Descrição: Esta classe encapsula um cliente CORBA.
// Ele deverá ser executado por um usuário
// que deseja efetuar uma pesquisa num documento XML
// remoto. Para rodar o cliente, o usuario
// deverá informar na linha de comando a consulta desejada (no formato pattern reduzido)
// e o nome de um arquivo onde o resultado da pesquisa deverá ser gravado.
// A linguagem de consulta é baseada na linguagem pattern, usada nos arquivos de estilo XSL
// Ela é parecida com o formato usado para informar caminhos de diretórios no Windows ou no
// MS-DOS. Consiste em nomear os nós desejados separados por barras (/). Exemplos:
// biblioteca/livro/autor --> irá retornar um documento XML contendo os elementos "autor"
//           //contidos em elementos "livro" contidos no elemento // "biblioteca"
// plantas/flores--> irá retornar um documento XML contendo
// os elementos
// "flores" que estiverem contidos em elementos "plantas"
// O símbolo # pode ser usado para designar todos os
// elementos de um determinado nível.
// Exemplo:
// biblioteca/livro/# --> irá retornar um documento XML
// contendo todos os elementos
```

A APLICAÇÃO CLIENTE

```
//que pertencem aos elementos "livro" que pertencem ao
//elemento "biblioteca".
public class Cliente {

//Rotina...: main
//Entradas.:String[] (java.lang)
//Saídas...: void
//Descrição: Esta rotina deve executar os procedimentos
//necessários para efetuar a
//consulta especificada pelo usuário e gravá-la num
//arquivo.
public static void main(String[] args) {

// Inicializa o ORB
String[] inic = {" "};
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(inic ,null);

// Cria um ID que irá identificar o objeto remoto
byte[] managerDocumentoXML =
        "DocumentoXML".getBytes();

// Localiza e efetua uma ligação com um objeto //servidor
XMLQuery.DocumentoXML DocumentoXMLLocal =
XMLQuery.DocumentoXMLHelper.bind(orb,"/DocumentoXML_poa",
        managerDocumentoXML);

// Executa o método que irá efetuar a busca no objeto remoto. O resultado será
// gravado na String Resultado.
String Resultado = new String("");

long tempo_inicial = System.currentTimeMillis();

Resultado=DocumentoXMLLocal.ObtemColecaoXML(args[0]);

long tempo_final = System.currentTimeMillis();
System.out.println("media do tempo gasto="
```

A APLICAÇÃO CLIENTE

```

+ ((tempo_final -
      tempo_inicial)/1000f)+"msecs");

// Grava resultados no arquivo especificado na linha de comandos. Caso ocorra algum
// erro durante a gravação, uma mensagem de erro é exibida para alertar o usuário
// do fato.
try {

    FileWriter fw = new FileWriter(args[1]);
    fw.write(Resultado);
    fw.flush();
    fw.close();

} catch (Exception e) {

    System.out.println
    ("Não consegui criar arquivo.");

};
}
}

```

QUADRO 11: LISTAGEM DO CÓDIGO DA APLICAÇÃO CLIENTE

A APLICAÇÃO SERVIDORA

```

// Arquivo : Servidor.java
// Autor   : Ivonei Freitas da Silva
// Local/Data: Cascavel, 2000
// Classes : Servidor

import org.omg.PortableServer.*;

// Classe: Servidor

```

A APLICAÇÃO SERVIDORA

```

// Variáveis públicas: nenhuma
// Rotinas públicas : void main(String[])
// Descrição: Esta classe encapsula um servidor CORBA. Ele deverá ser startado e deixado
// rodando indefinidamente na memória junto com o agente inteligente CORBA (osagent).
// Para startar o servidor, o usuário deverá informar o nome do arquivo XML que será
// parseado e disponibilizado para consultas. Esta classe interagirá com a classe
// DocumentoXMLImpl para parsear e consultar o arquivo XML.

public class Servidor {

    // Rotina...: main
    // Entradas.: String[] (java.lang)
    // Saídas...: void
    // Descrição: Esta rotina deve interagir com o ORB e deixar uma cópia de um objeto do tipo
    // DocumentoXMLImpl que será acessada remotamente pelos clientes CORBA.
    public static void main(String[] args) {
        try {

            // Inicializa o ORB
            String[] inic = {" "};
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(inic ,null);

            // Obtém uma referência ao POA padrão para que possa criar um POA próprio
            POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

            // Cria as políticas para o nosso POA
            org.omg.CORBA.Policy[] policies = {
                rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
            };

            // Cria o POA que interagirá com o ORB
            POA myPOA = rootPOA.create_POA( "DocumentoXML_poa", rootPOA.the_POAManager(),
                policies );

            // Cria o objeto que será disponibilizado na rede
            DocumentoXMLImpl myDocumentoXMLImpl = new DocumentoXMLImpl(args[0]);

```

A APLICAÇÃO SERVIDORA

```

// Cria um ID para identificar o objeto
byte[] managerDocumentoXML = "DocumentoXML".getBytes();

// Ativa o servidor
myPOA.activate_object_with_id(managerDocumentoXML, myDocumentoXMLImpl);

// Ativar o gerenciador de POA's
rootPOA.the_POAManager().activate();

// Exibe mensagem informando que estamos prontos para receber pedidos remotos
System.out.println("Aguardando pedidos!");

// Aguarda os pedidos
orb.run();
}
catch (Exception e) {
    e.printStackTrace();
}
}
}

```

QUADRO 12: LISTAGEM DO CÓDIGO DA APLICAÇÃO SERVIDORA

A CLASSE DocumentoXMLImpl

```

// Arquivo : DocumentoXMLImpl.java
// Autor : Ivonei Freitas
// Local/Data: Cascavel, 2000
// Classes : DocumentoXMLImpl
//      TToken
//      Tokeniza

import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import javax.XML.parsers.*;
import org.XML.sax.*;

```


A CLASSE DocumentoXMLImpl

```

// Classe: DocumentoXMLImpl
// Variáveis públicas: nenhuma
// Rotinas públicas : construtor DocumentoXMLImpl(String)
//      String  ObtemColecaoXML (String)
// Descrição: Esta é a classe principal do arquivo. Deverá ser disponibilizada pelo servidor
// CORBA para uso remoto pelos clientes CORBA. Seu objetivo é gerar uma estrutura em forma
// de árvore que representará um determinado arquivo XML. Esta estrutura deverá ser
// consultada pelas aplicações cliente e gerará resultados em forma de arquivos XML.
// Esta classe encapsula um arquivo XML que pode ser consultado usando uma sintaxe parecida
// com os patterns (modelos), usados em documentos de estilo (XSL).
public class DocumentoXMLImpl extends XMLQuery.DocumentoXMLPOA{
    static int ContMostraArvore=0;
    static DocumentBuilderFactory dbf;
    static DocumentBuilder db;
    static Document DocumentoOrigem ,
    DocumentoDestino;
    static String Resultado = new String("");
    // Rotina...: MostraArvore
    // Entradas.: Node (org.w3c.dom)
    // Saídas...: void
    // Descrição: Exibir toda a árvore de um nó na forma textual na tela. O algoritmo consiste
    // em tentar exibir todo um nível da árvore de nós. Caso um destes nós tenha filhos
    // a função utiliza uma chamada recursiva para processar os nós filhos.
    static void MostraArvore(Node No){
        // Mostra todo este nível da árvore de objetos.
        // A variável No sempre recebe o proximo objeto deste nivel
        while(No!=null){
            //Esta parte escreve o nome do elemento
            try{
                // Esta parte insere um espaço antes do nome do elemento.
                // Este espaço é baseado na variável estática ContMostraArvore
                // (definida fora do método )que é incrementada
                // sempre que esta rotina é chamada novamente (sempre que o
                // programa descer um nível na árvore de objetos)
                for(int i=0;i<=ContMostraArvore;i++){System.out.print(" ");}
                if(No.getNodeType()==No.ELEMENT_NODE) // constante definida pela api.
                    System.out.println("Elemento:<" +No.getNodeName()+">");
            }
        }
    }
}

```

A CLASSE DocumentoXMLImpl

```

else
    System.out.println("Texto:"+No.getNodeValue());
}catch(Exception e){System.out.println("Ocorreu um erro");};
// Se o nó tiver filhos (outro nível), chamada recursiva.
// Incrementa ContMostraArvore para gerar uma camada níveis na tela.
if(No.hasChildNodes()){
    ContMostraArvore++;
    MostraArvore(No.getFirstChild());
    ContMostraArvore--;
}
// Passa para próximo nó no mesmo nível

No=No.getNextSibling();
}
}
// Rotina...: FiltraArvore
// Entradas.: Node (org.w3c.dom), LinkedList (java.util), int
// Saídas...: void
// Descrição: Filtrar uma árvore de nós baseada numa consulta que é fornecida num
// argumento da chamada. O resultado será colocada na String Resultado (variável global
// da classe). O algoritmo é idêntico à função MostraArvore, sendo geradas chamadas
// recursivas para o processamento de nós filhos.
static void FiltraArvore(Node No, LinkedList Consulta, int Pos){
    boolean Flag = true;
    // O while é usado para processar todos os nós deste nível.
    while(No!=null){
        // Este aglomerado de if's filtra realmente o nó atual (decide se ele deve ser
        // transferido para a variável Resultado). Basicamente, um nó só deve ser
        // transferido se ele for um elemento cujo nome é igual àquele definido na
        // consulta ou se a consulta for = # (Código de token=3) ou ainda, se ele
        // for um nó do tipo textual.

        if(No.getNodeType()==No.ELEMENT_NODE){

            if (No.getNodeName().equalsIgnoreCase(((TToken)(Consulta.get(Pos))).Valor)){
                Flag=true;
            }else{
                if (((TToken)(Consulta.get(Pos))).Token==3){

```


A CLASSE DocumentoXMLImpl

```

java.io.IOException{
// Cria os objetos necessário para parsear um arquivo XML.

    dbf = DocumentBuilderFactory.newInstance();
    db = dbf.newDocumentBuilder();

    // Abre o arquivo desejado
    File f = new File(NomeArquivo);
    try{
        // Chama a rotina parse para criar a árvore de nós desejada que será armazenada
        // na variável global DocumentoOrigem
        DocumentoOrigem = db.parse (f);
        DocumentoOrigem.getDocumentElement().normalize ();
        // Exibe todo o documento na tela.
        MostraArvore(DocumentoOrigem.getDocumentElement());
    }catch(Exception e){}
    }
// Rotina...: ObtemColecaoXML
// Entradas.: String
// Saídas...: String
// Descrição: Esta rotina irá receber uma consulta vinda do usuário, processá-la e retornar
// o documento XML que resultar desta pesquisa em uma String.
    public String ObtemColecaoXML(String Pattern){
        // Chama a rotina ObtemTokens (definida na classe Tokeniza, logo abaixo) para criar
        // uma estrutura dos tokens da consulta na forma de uma lista ligada, que será mais
        // cômoda para leitura pelas rotinas de filtragem desenvolvidas.

        LinkedList Consulta=Tokeniza.ObtemTokens(Pattern, 0);
        // Zera a variável Resultado para ter certeza de que vários arquivos XML não sejam
        // concatenados nesta variável.

        Resultado="" ";
        // Utiliza a rotina FiltraArvore definida acima para gerar o Resultado esperado pelo
        // usuário
        FiltraArvore(DocumentoOrigem.getDocumentElement(), Consulta, 0);
        // Retorna o documento XML contido na variável Resultado.
        return(Resultado);
    }

```

A CLASSE DocumentoXMLImpl

```

}
// Classe: TToken
// Variáveis públicas: int Token -> Código de um token de consulta do usuário: 2 para
//
//          qualquer caracter válido e 3 para #.
//          String Valor -> Valor do token. Pode ser # para o token 3 ou a própria
//          cadeia de caracteres token para o token 2.
// Rotinas públicas : construtor TToken(int, String)
// Descrição: Cria uma estrutura para armazenar Tokens que definirão a consulta do usuário

class TToken{
    public int Token;
    public String Valor;
    // Rotina...: TToken (construtor)
    // Entradas.: int, String (java.lang)
    // Saídas...: N/A
    // Descrição: Um construtor que inicializará as variáveis públicas da classe.
    TToken(int i, String s){
        Token=i;
        Valor=new String(s);
    }
}

// Classe: Tokeniza
// Variáveis públicas: nenhuma
// Rotinas públicas : LinkedList ObtemTokens(String, int)
// Descrição: Esta classe foi criada para encapsular rotinas de geração de uma lista ligada
// que facilitará a manipulação da consulta do usuário.

class Tokeniza{
    // A variável Tokens conterá a lista ligada que será retornada ao usuário
    static LinkedList Tokens;

    // Rotina...: EhCaracterValido
    // Entradas.: char
    // Saídas...: boolean
    // Descrição: Simplesmente recebe um caracter e retorna true se este é um caracter válido
    // ou false caso contrário. Como caracter válido compreende-se as letras (maiúsculas ou

```

A CLASSE DocumentoXMLImpl

```

// minúsculas), os algarismos (de 0 a 9), "underscore" ( _ ), hífen (-) ou dois pontos (:)
static boolean EhCaracterValido(char c){
    return( ((c>='a') && (c<='z')) ||
            ((c>='A') && (c<='Z')) ||
            ((c>='0') && (c<='9')) ||
            (c=='_') || (c=='-') || (c=='.'));
}
// Rotina...: ObtemTokens
// Entradas.: String (java.lang), int
// Saídas...: LinkedList (java.util)
// Descrição: Varre a consulta fornecida pelo usuário e constrói uma lista ligada contendo
// os tokens que representarão esta consulta, através da estrutura TToken. Esta rotina
// usa recursividade para varrer a consulta. Existem apenas dois tokens válidos: 2 para
// caracteres válidos e 3 para # (que representa todos os nós do nível). Exemplo de
// consultas e seus correspondentes tokens: book/orders/##/title (tokens: 2,2,3,2).

public static LinkedList ObtemTokens(String Consulta, int Pos){

    if (Pos==0){
        Tokens = new LinkedList();
    }
    // Se o caracter analisado for uma barra simples, simplesmente incrementa a variável
    // Pos para passar para a posição seguinte.
    if(Consulta.charAt(Pos)=='/'){
        Pos++;
    }
    // Se o caracter analisado for sustenido (#), acrescenta um nó à lista ligada. Este nó
    // conterá uma estrutura TToken que será composta por um número 3 e o valor #.
    }else if(Consulta.charAt(Pos)=='#'){
        Tokens.add(new TToken(3, "#"));
        Pos++;
    }
    // Se o caracter em questão for um caracter válido (ver discussão acerca de caracteres
    // válidos na descrição da rotina EhCaracterValido) continua varrendo a String até
    // encontrar um caracter não válido (este conjunto de caracteres válidos será o nosso
    // token e representará um nó num determinado nível). Depois acrescenta este token
    // à lista ligada de tokens através da estrutura TToken que será composta do número
    // 2 e do conjunto de caracteres válidos obtido.
    }else if(EhCaracterValido(Consulta.charAt(Pos))){
        String wToken = new String("");

```

A CLASSE DocumentoXMLImpl

```
try {
    while(EhCaracterValido(Consulta.charAt(Pos)))
        wToken=wToken+Consulta.charAt(Pos++);
    }catch(Exception e){};
    Tokens.add(new TToken(2, wToken));
    // Caso o caracter analisado não seja uma barra (/) ou um suspenido (#) ou um caracter
    // válido, um erro estará configurado. Exibe mensagem de erro informando o fato ao
    // usuário.
    }else{
        System.out.println("Erro na posição " +Pos);
        Pos++;
    };
    // Se a próxima posição a ser analisada (variável Pos) for menor que o tamanho da consulta
    // executa chamada recursiva para continuar analisando esta consulta. Caso contrário,
    // retorna a lista obtida.
    if(Pos<Consulta.length())
        ObtemTokens(Consulta, Pos);
    return(Tokens);
}
}
```

QUADRO 13: LISTAGEM DO CÓDIGO DA CLASSE QUE IMPLEMENTA O SERVIÇO PESQUISA/FILTRAGEM DE DADOS XML

ANEXO 3: QUADRO QUE LISTA O CONTEÚDO DO ARQUIVO XML UTILIZADO PELA APLICAÇÃO

Arquivo **book-order.XML**, quadro 14, utilizado para testar a funcionalidade da aplicação e também para os testes de desempenho da mesma.

book-order.XML

```
<?XML version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<!DOCTYPE Order [

<!--
  A DTD for a real on-line order system wouldn't look very much like
  this, though it would be natural for such systems to use valid XML.

  Note also that when this is processed, the DTD is discarded by DOM.
  In this case, that means that the default namespace for the "Order"
  will also be discarded.

  To let successive nodes in a series of XML processors work without
  losing data, you need to work around that "DTD Discarding" problem.
  One way is not to use DTDs. A better alternative would be to stick
  to a single external DTD and re-insert it in each processing stage.

  That is, the DOCTYPE declaration would have no internal subset and
  would look like "<!DOCTYPE Order SYSTEM 'http://www....!>".
-->

<!ELEMENT Order (Customer,Manifest,Receipt)>
<!ATTLIST Order XMLns CDATA #FIXED "http://www.example.com/myschema.XML">

<!ELEMENT Customer (Name, Cardnum)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Cardnum (#PCDATA)>

<!ELEMENT Manifest (Item*)>

<!ELEMENT Item (ID,Title,Quantity,UnitPrice)>
```


book-order.XML

```

<!ELEMENT ID (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT UnitPrice (#PCDATA)>

<!ELEMENT Receipt (Subtotal,Tax,Total)>
<!ELEMENT Subtotal (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ELEMENT Total (#PCDATA)>

]>

<Order>
  <Customer>
    <Name>Bill Buckram</Name>
    <Cardnum>234 234 234 234</Cardnum>
  </Customer>
  <Manifest>
    <Item>
      <ID>209</ID>
      <Title>
        Duke: A Biography of the Java Evangelist
      </Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
    <Item>
      <ID>208</ID>
      <Title>
        100% Pure: Making Cross Platform Deployment a Reality
      </Title>
      <Quantity>1</Quantity>
      <UnitPrice>$10.75</UnitPrice>
    </Item>
    <Item>
      <ID>204</ID>
      <Title>
        Making the Transition from C++ to the Java(tm) Language

```

book-order.XML

```

</Title>
  <Quantity>1</Quantity>
  <UnitPrice>$10.75</UnitPrice>
</Item>
<Item>
  <ID>202</ID>
  <Title>Web Servers for Fun and Profit</Title>
  <Quantity>1</Quantity>
  <UnitPrice>$10.75</UnitPrice>
</Item>
<Item>
  <ID>210</ID>
  <Title>
    I Think Not: Dukes Likeness to the Federation Insignia
  </Title>
  <Quantity>1</Quantity>
  <UnitPrice>$10.75</UnitPrice>
</Item>
</Manifest>
<Receipt>
  <Subtotal>$53.75</Subtotal>
  <Tax>$4.43</Tax>
  <Total>$58.18</Total>
</Receipt>
</Order>

```

QUADRO 14: LISTAGEM DO ARQUIVO BOOK-ORDER.XML

ANEXO 4: TABELAS DE MENSURAÇÃO DOS TEMPOS DE RESPOSTA DAS APLICAÇÕES

Tabelas com os tempos de resposta das aplicações. Tabela 8 e 9.

TABELA 8: TEMPOS DE RESPOSTA COM ARQUIVOS GRANDES

Aplicação XML –280.051 bytes (arquivo)	Aplicação HTML –21.190 bytes (arquivo)
(ms)	(ms)
1.32 (primeira medida)	1.65 (primeira medida)
1.05	1.26
0.99	1.21
0.99	1.15
0.99	1.21
0.99	1.21
0.99	1.21
0.99	1.15
0.99	1.15
1.04	1.15

TABELA 9: TEMPOS DE RESPOSTA COM ARQUIVOS PEQUENOS

Aplicação XML – 2.522 bytes (arquivo)	Aplicação HTML – 2.168 bytes (arquivo)
(ms)	(ms)
1.26 (primeira medida)	1.43 (primeira medida)
0.98	0.99
0.98	0.98
0.98	0.99
0.98	1.00
0.98	0.99
0.99	0.99
0.98	0.98
0.98	0.99
0.98	0.99
0.98	0.99