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

Fabio Alexandre Spanhol

Uma Aplicação de XML para Auxiliar na Gerência de Redes

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. Carlos Becker Westphall, Orientador

Florianópolis, dezembro de 2002

Uma Aplicação de XML para Auxiliar na Gerência de Redes

Fabio Alexandre Spanhol

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. Ostuni Gauthier, Dr.
Coordenador

Banca Examinadora:

Prof. Carlos Becker Westphall, Dr.
Presidente

Profª. Carla Merkle Westphall, Drª.

Prof. Mário Antonio Ribeiro Dantas, Dr.

Prof. Roberto Willrich, Dr.

“Para ser grande, sê inteiro: nada
Teu exagera ou exclui.
Sê todo em cada coisa. Põe quanto és
No mínimo que fazes.
Assim em cada lago a lua toda
Brilha, porque alta vive”.

(Ricardo Reis, um dos heterônimos de Fernando Pessoa)

”Estudai como se vivêsseis para sempre, vivei como se fôsseis morrer amanhã”
(Santo Isidoro De Sevilha)

Aos meus queridos pais Alcino Spanhol e Deisy do Carmo Bahú Spanhol, por todos os anos de apoio, dedicação e esmero; mostrando-me sempre como superar de maneira digna as mesquinharias e injustiças que permeiam nossa efêmera peregrinação por este mundo.

À minha amada esposa Claudiana Soerensen e ao meu precioso filho Marcus Vinícius Soerensen Spanhol, que constituíram minha fortaleza em suas singelas fragilidades.

AGRADECIMENTOS

Ao Criador Supremo, onipresente e onisciente, simultaneamente particular e cósmico. Inspiração e fonte primeira de todo conhecimento.

Carinhosamente à minha amada esposa, Claudiana Soerensen, sempre amorosa companheira, pelas preciosas horas de nosso convívio tolhidas pelo presente trabalho.

A Marcus Vinícius Soerensen Spanhol, o Vini, meu filho amado. Um pequenino astro que trouxe aos meus dias um novo ânimo. Peço perdão pelas minhas muitas ausências, tanto físicas como mentais. Espero poder despender mais tempo para viver e compartilhar esses preciosos e únicos momentos de sua infância.

Aos meus pais, Alcino Spanhol e Deisy do Carmo Bahú Spanhol, pelo modelo de retidão de caráter transmitido desde os meus tenros anos.

Aos meus irmãos, Ana Cláudia Spanhol, Márcio Rogério Spanhol e Silvia Letícia Spanhol, pelo apoio em muitos pequenos embaraços do cotidiano.

Especialmente ao meu orientador, professor Carlos Becker Westphall, que me acolheu em um momento bastante conturbado do curso e incentivou-me a concluir esse trabalho.

A professora Carla Merkle Westphall e Cleverson Alessandro Veronez, membros da banca do meu Trabalho Individual, pelas observações e apontamentos inestimáveis para a conclusão deste trabalho.

Aos colegas da pós-graduação Wagner Tatsuya Watanabe e Luis Marco Cáceres Alvarez, por mostrarem-se em inúmeras ocasiões não apenas colegas, mas verdadeiros amigos.

Ao meu colega de graduação e amigo Jefferson Gustavo Martins, por suportar-me como ”hóspede regular” do seu apartamento em Florianópolis.

Ao amigo e confidente de muitas horas Ivonei Freitas da Silva, pelos muitos “conselhos” e direcionamentos.

Ao grande amigo professor Ivanor Luis Guarnieri, pela contagiante sede de conhecimento, sem dúvida um inspirador e escasso modelo de intelectual.

Aos bons professores do curso de Pós-Graduação em Ciência da Computação da UFSC, por constituírem um verdadeiro modelo de profissional que sempre tento (e tentarei) imitar em minha atuação diária. Aprendi muito com vocês nesse curto espaço de tempo e algumas empolgantes aulas ficaram registradas firmemente em minha memória.

A Vera Lúcia Sodré Teixeira, a Verinha, secretária da Pós-Graduação, pela sua simpatia e atenção habituais.

RESUMO

XML (eX*tensible Markup Language*) apresenta como principais características a distinção entre interface, processos e dados; provendo flexibilização no intercâmbio de dados; customização de linguagens de marcação; autodescrição de dados, estruturação e integração dos dados. Essa dissociação entre a estrutura da apresentação e o conteúdo do documento, permite que um mesmo documento possa ser apresentado em diferentes formatos.

Recentemente, o emprego de XML para definir modelos de informação de gerenciamento e processar informação nas aplicações de gerência vem se tornando atrativo.

Nesse contexto, este trabalho apresenta uma aplicação de XML no modelo de gerenciamento SNMP, como forma de produzir documentos de informação de gerência que sejam estruturados, consistentes, autodescritivos e possam ser facilmente apresentados em formatos diversos, podendo também alimentar outros sistemas de informação.

ABSTRACT

XML (eXtensible Markup Language) presents as main characteristics the distinction between interface, processes and data; providing flexibility on data interchange; customization of marking languages; self-describle of data, structuration and integration of the data. This disjoin between the structure of the presentation and the content of the document, allows that one exactly document can be presented in different formats.

Recently, the qpplication of XML to define models of management information and to process information in the management applications is becomes attractive.

In this context, this work presents the application of XML in SNMP management model, for produce management information documents structuralized, consistents, self-describle and can easily be showed in diverse formats, also being able to feed other information systems.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

ASCII – *American Standard Code for Information Interchange*

ASN.1 – *Abstract Syntax Notation .1*

ASP – *Active Server Pages*

CERN – *Centro*

CGI – *Common Gateway Interface*

CMIP – *Commom Management Information Protocol*

CORBA – *Commom Obeject Request Broker Architecture*

CSS – *Cascading Style Sheets*

DoD – *Department of Defense*

DOM – *Document Object Model*

DII – *Dynamic Invocation Interface*

DSI – *Dynamic Skeleton Interface*

DSSSL – *Document Style Semantics and Specification Language*

DTD – *Document Type Definition*

ESIOP *Environment-Specific Inter-ORB Protocol*

EWS *Embedded Web Server*

GI – *Generic Identifier*

GIOP – *Generic Inter-ORB Protocol*

HTML – *Hyper Text Markup Language*

IAB – *Internet Activities Board*

IBM – *International Business Machines*

IDL – *Interface Definition Language*

IEEE – *Institute of Electical and Electronics Engineers*

IIOP – *Internet Inter-ORB Protocol*

INRIA – *Instituit National de Recherche em Informatique et em Automatique*

IP – *Internet Protocol*

ISO – *International Organization for Standardization*

JIT – *Just In-Time*

MIB – *Management Information Base*

MIT – *Massachussets Institut Of Technology*

MS – *Microsoft Corporation*

OID – *Object Identifier*

OMG – *Object Management Group*

ORB – *Object Request Broker*

OSF – *Open Software Foundation*

OSI – *Open Systems Interconection*

PDU – *Protocol Data Unit*

POA – *Portable Object Adapter*

PI – *Processing Instructions*

RFC – *Request For Comments*

RTF – *Rich Text Format*

SAX – *Simple Application programming interface for Xml*

SNMP – *Simple Network Management Protocol*

SGML – *Standard Generalized Markup Language*

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

UFSC – *Universidade Federal de Santa Catarina*

UNIOESTE – *Universidade Estadual do Oeste do Paraná*

W3C – *World Wide Web Consortium*

WWW – *World Wide Web*

XML – *Extensible Markup Language*

XSL – *Extensible Style Language*

XSL – *XSL Transformations*

1. Introdução

As redes de computadores foram inicialmente concebidas como um meio de compartilhar dispositivos e periféricos mais dispendiosos como impressoras *laser*, *modems* velozes e outros. Contudo, os benefícios proporcionados por estas redes estimularam uma expansão crescente, e ainda contínua, das mesmas nos mais diversos segmentos da atividade humana, relegando assim o seu objetivo inicial (o simples compartilhamento de dispositivos) a um plano de importância secundária. Essa expansão demanda manutenção e planejamento futuro, a denominada gerência de redes.

O gerenciamento de redes de computadores tem como objetivos controlar, administrar e monitorar eficientemente os recursos de *hardware* e *software* em um ambiente computacionalmente distribuído. Não se consegue gerenciar uma rede com esforços humanos isolados. A complexidade de uma rede, devido à quantidade e heterogeneidade de equipamentos e fornecedores, exige o uso de automação no gerenciamento de rede [STALLINGS 00]. A velocidade com que o estado da rede varia, a necessidade de constância no monitoramento e a necessidade de executar ações de controle e manutenção implicam na utilização de um sistema computacional dinâmico de supervisão, as chamadas aplicações de gerência. Tais aplicações requerem interfaces padronizadas para trocar informação entre os sistemas de gerenciamento, possuindo capacidade de extensão para tratar mudanças de forma rápida e fornecimento de meios para o gerenciamento de grandes redes [HAGGERTY 98].

Mais recentemente, a Internet expandiu-se, extrapolando os domínios de aplicações acadêmicas, militares e de pesquisa para inserir-se no âmbito comercial/corporativo e até mesmo doméstico. Considerando a disseminação das redes e da Internet, constata-se que é cada vez mais freqüente a utilização de aplicações sobre a arquitetura TCP/IP pela grande maioria dos usuários de tais redes. Nesse aspecto, sistemas de informação estão se integrando com servidores *Web*[1] com o objetivo de transformar consultas realizadas por usuários em resultados a serem exibidos em *browsers Web* [DEROSE 99], [GRAHAM 00]. Assim, a Internet, notadamente a *Web*,

[1] Software de servidor que utiliza o protocolo HTTP para fornecer documentos HTML e quaisquer arquivos e *scripts* associados quando solicitados por uma aplicação cliente, como um *browser Web*. A conexão entre o cliente e o servidor é geralmente rompida após o fornecimento do documento ou arquivo solicitado. Os servidores HTTP são usados em *sites* da Web e de intranets.

está se configurando numa infra-estrutura para distribuição de informação essencial para qualquer organização [THOMPSON 98] [BERG 99]. E uma das possibilidades proporcionadas pela *Web* é o acesso e tratamento de informações de gerenciamento de redes [BARILLAUD 97].

Todavia, as pessoas e organizações que disponibilizam informações na rede estruturam essas informações de maneiras particulares. Estruturações distintas da informação não causam problemas enquanto tais informações são empregadas de forma isolada, pelo indivíduo ou organização que as gerou. Porém, na medida em que a informação precisa ser compartilhada com outras pessoas ou organizações, a carência de uma estrutura padronizada acarreta empecilhos, principalmente a incompatibilidade entre o sistema receptor e o sistema emissor da informação.

Somando-se a essa considerável disseminação de microcomputadores e estações de trabalho, o aumento nas suas capacidades de processamento, e o surgimento de redes de comunicação com grande largura de banda, nota-se o desenvolvimento cada vez maior de aplicações distribuídas. A importância de sistemas distribuídos também tem crescido devido, principalmente, as tendências organizacionais tais como *downsizing*[2], que demandam o intercâmbio de informação dentro da própria organização e entre organizações cooperantes.

Os sistemas distribuídos apresentam vantagens advindas da distribuição tais como disponibilidade, desempenho, otimização de custos, dentre outras. Entretanto, apresentam também as características de afastamento, concorrência, falta de estado global, ocorrência de falhas parciais, assincronismo, heterogeneidade, autonomia, evolução e mobilidade [ISO 95].

Diversos modelos e arquiteturas distribuídas têm sido propostos, oferecendo conceitos e implementações que auxiliam no tratamento das características de distribuição citadas anteriormente. Notadamente a característica de heterogeneidade impõe a necessidade de especificações abertas, com interfaces padronizadas e públicas, levando ao desenvolvimento de *middlewares* abertos. De modo geral, um *middleware*

[2] É a migração de sistemas de grande porte/centralizados (*mainframe*) para a computação distribuída, redes no esquema Cliente/Servidor, ou seu gerenciamento (já não possuem servidor)

pode ser definido como sendo uma camada de software, residente acima do sistema operacional e do substrato de comunicação, que oferece abstrações de alto nível, com objetivo de facilitar a codificação de aplicações distribuídas. As abstrações disponibilizadas fornecem uma visão uniforme na utilização de recursos heterogêneos existentes nas camadas de sistema operacional e redes.

O conceito de sistemas abertos aborda um significante leque de tecnologias e especificações, envolvendo a questão de interoperabilidade de uma variedade de sistemas baseados em padrões formais ou de fato. De forma diversa dos ambientes proprietários, os ambientes de sistemas abertos permitem aos usuários optarem dentre um vasto grupo de computadores e tecnologias, quais se adequam às suas necessidades. Esses sistemas oferecem também uma crescente interoperabilidade, escalabilidade e portabilidade, permitindo sistemas distintos conviverem juntos, e o software implementado em uma plataforma específica ser executado em outra com mínimas adaptações.

Apesar de inúmeras vantagens na sua utilização, o advento de sistemas abertos torna mais complexo o processo de desenvolvimento de novas tecnologias e adesão às especificações abertas. A solução dessa problemática, preservando as vantagens da computação heterogênea, tem levado ao surgimento de diversas organizações e consórcios tais como OMG (*Object Management Group*)[3], X/OPEN[4] e OSF (*Open Software Foundation*)[5].

A organização OMG estabeleceu a arquitetura CORBA (*Common Object Request Broker Architecture*) [OMG 01] como uma forma de especificar um *middleware* aberto composto de objetos distribuídos. CORBA define um tipo de "barramento de software" permitindo que componentes de software sejam acoplados, formando um sistema coeso. Os conceitos utilizados nos componentes de software são similares aos dos componentes de hardware, e são implementados na arquitetura CORBA através do paradigma da orientação a objetos.

[3] *www.omg.org*
[4] *www.opengroup.org*
[5] *www.osf.org*

Nessa conjuntura, CORBA já vem sendo utilizada sobre a infraestrutura *Web* [BARILLAUD 97] como uma ferramenta para permitir a implementação de um sistema de gerenciamento de rede distribuído [BAROTTO 98], [BAROTTO 00] entre as máquinas que compõem a rede, independente de plataforma e com mecanismo de instalação automática, visto que estará disponível através de *browsers Web*.

Neste trabalho, é discutida a adoção da tecnologia XML (*eXtensible Markup Language*), especificamente em sistemas de gerenciamento de redes, propondo uma flexibilização e estruturação no intercâmbio de informações gerenciais. XML é uma metalinguagem que possibilita a definição de linguagens de marcação personalizadas, permitindo a especificação da sintaxe e a validação de documentos nessa linguagem personalizada [CASTRO 01], chamada aplicação XML. A abordagem de XML, com clara separação entre os dados, a estrutura e a maneira como serão apresentados tais dados, permite que sejam implementados programas que extraiam dados de documentos XML, transforme-os, unifique-os a outras fontes e disponibilize-os para apresentação em vários formatos distintos.

As características de XML tornam-na uma ferramenta ideal para representar dados, apoiando na definição de um modelo de conteúdo padronizado, em uma plataforma aberta, independente de fabricantes e em uma linguagem neutra [JU 02].

Focando a gerência de redes, o emprego de XML para definir modelos de informação de gerenciamento e processar informação nas aplicações de gerência vem se tornando atrativo [LEWIS 01]. Nesse aspecto, XML e suas tecnologias correlatas apresentam vantagens como [MARTIN-FLATIN 00]:

- melhor integração dos dados gerenciados
- ligação mais flexível entre o objeto gerenciado e aplicação gerente;
- interoperabilidade entre aplicações de gerenciamento de diferentes fabricantes;
- apresentação das informações gerenciais facilidade e estendida para uma ampla variedade de formatos;

- possibilidade de transformação automática das informações de gerenciamento originais, agregando valor decisório nas novas informações produzidas;

- validação dos dados de gerenciamento automática e centralizada.

Para demonstrar a possibilidade do uso de XML na gerência de redes, será considerado um ambiente de rede existente na Unioeste (Universidade Estadual do Oeste do Paraná), Paraná. Para tanto, será implementado um aplicativo de gerenciamento que extraia informações gerenciais de um dispositivo de rede dotado de agente SNMP. O aplicativo deve comunicar-se com o agente SNMP do dispositivo através de consultas SNMP e gerar documentos XML a partir das respostas recebidas. Tais documentos poderão ser processados por ferramentas XML para serem exportados/apresentados em diferentes formatos.

Também, pretende-se sugerir um modelo para unir a arquitetura CORBA, o padrão de gerenciamento SNMP e as tecnologias XML.

1.1 Objetivos

Nessa seção serão apresentados os objetivos que este trabalho pretende atingir.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é apresentar um estudo da aplicação de tecnologias de distribuição e interoperação de sistemas utilizando XML para apresentação de informações gerenciais em *browsers Web*. Para proporcionar o estudo será implementado um protótipo de um sistema distribuído destinado ao gerenciamento de dispositivos conectados em um ambiente de rede operando sob a arquitetura TCP/IP. Tal sistema de gerenciamento deve suportar o protocolo de gerência SNMP e estruturar-se de maneira distribuída. Também será proposto um modelo de arquitetura que se apóie

no modelo CORBA/IIOP para prover objetos distribuídos e agentes de gerência multiplataforma.

1.1.2 Objetivos Específicos

A fim de atingir a meta descrita no objetivo geral, foram estipulados os seguintes objetivos específicos:

- Estudo dos principais padrões e ferramentas relacionados com a tecnologia *Web*, enfocando a integração de SNMP e CORBA com tal tecnologia para prover um ambiente de gerenciamento de redes de computadores acessível via *Web*;
- implementação de uma aplicação que extraia informações das MIBs, utilizando o protocolo SNMP e gere documentos XML;
- implementação de um protótipo que demonstre a manipulação dos documentos XML, construindo páginas XML que possam ser visualizadas pelos administradores de rede em *browsers Web* compatíveis com XML, possibilitando um acesso remoto às informações de gerência filtradas, sumarizadas e tratadas;
- teste e avaliação dos resultados obtidos, estabelecendo um panorama de perspectivas futuras para aplicação integrada das tecnologias SNMP, XML e CORBA para prover gerenciamento distribuído de redes.

1.2 Trabalhos Correlatos

Em [BAROTTO 98] e [BAROTTO 00] é apresentada a implementação de um sistema distribuído de gerenciamento aplicado a um *cluster*[6] de estações de alta performance existente na UFSC (Universidade Federal de Santa Catarina). Tal sistema

[6] conjunto de estações

opera sobre os protocolos SNMP e IIOP/CORBA, utilizando *browsers Web* como interface com o usuário final.

Uma nova arquitetura de gerenciamento é apresentada em [JU 02], combinando EWS (*Embedded Web Server*) com XML, DOM (*Document Object Model*) e *XPath* para unificar elementos gerenciados com o gerenciamento de redes. A tecnologia XML é usada para modelar a informação de gerenciamento e a comunicação gerente-agente. A arquitetura proposta pressupõe a disponibilidade de EWS nos dispositivos gerenciados. Um EWS pode ser caracterizado como sendo um servidor *Web* embutido em um dispositivo de rede com propósito de gerenciamento, provendo aos usuários uma interface de gerenciamento baseada nas tecnologias comuns em *browsers Web* [JU 01].

1.3 Estruturação do Trabalho

Este trabalho está organizado como segue:

- O capítulo 2 apresenta uma visão geral das linguagens de marcação, enfocando XML e tecnologias inter-relacionadas.
- O capítulo 3 descreve de forma sucinta a linguagem Java e a motivação para sua escolha como ferramenta de programação para o protótipo proposto.
- No capítulo 4 é descrita, de maneira não exaustiva, a arquitetura CORBA.
- No capítulo 5 é apresentada a API *AdventSNMP*, implementação Java de uma biblioteca de gerenciamento de redes que opera sobre o protocolo SNMP, padrão de gerenciamento em redes TCP/IP. O modelo de gerenciamento de redes SNMP também é apresentado nesse capítulo.
- O capítulo 6 apresenta o ambiente de rede estudado e o protótipo da aplicação para demonstrar um emprego de XML nas tarefas de gerência de redes. Também é discutido um possível uso integrado de CORBA para prover uma característica de distribuição ao gerenciamento de redes.

- Finalmente, o capítulo 7 descreve os resultados alcançados e os percalços enfrentados, avaliando as vantagens e desvantagens da utilização conjunta das tecnologias para o gerenciamento de redes. Propostas futuras também são discutidas, seguidas pelo referencial bibliográfico consultado.

2. Visão Geral de XML

2.1 Introdução: Marcação

A marcação originou-se, já há bastante tempo, na indústria gráfica. Essa técnica da editoração tradicional consistia em anotar na obra manuscrita um conjunto de instruções para o compositor [BOSAK 99]. Tais anotações, também manuscritas, denominavam-se marcação (*mark-up*) [MARCHAL 00], sendo uma etapa distinta, após a escrita e antecedendo a composição gráfica, da obra a ser impressa.

Similarmente, linguagens de marcação são mecanismos que permitem especificar a forma como um certo documento deve ser interpretado ou apresentado pelos mais diferentes tipos de software.

2.1.1 Marcação procedimental

Freqüentemente, processadores de texto requerem que o usuário especifique a aparência do documento que está sendo editado, como selecionar um certo trecho do texto para ser apresentado em uma determinada fonte gráfica em negrito ou ainda o posicionamento de um certo parágrafo na página do documento. Tais informações de formatação são denominadas marcação (*markup code*), e em geral são armazenadas juntamente com o documento [GRAHAM 00], [MARCHAL 00].

Do ponto de vista metodológico, a tarefa do usuário é identificar os elementos significativos do texto que devem receber a formatação e selecionar um conjunto de comandos que produzirá o resultado de formatação desejado. Análogo ao processo de editoração gráfica tradicional existe um conjunto de instruções de marcação anexado ao texto. Um exemplo típico de marcação procedimental encontra-se no formato RTF (*Rich Text Format*), desenvolvido pela *Microsoft Corporation* (MS) e suportado por softwares de várias outras empresas para intercâmbio de documentos. O RTF indica como um processador de texto deve apresentar um documento gravado em tal formato.

Na fig. 2.1a é mostrado um documento formatado em RTF. Já na fig. 2.1b é apresentada a correspondente exibição do documento no processador de textos MS-Word 2000.

```
{\rtf1\ansi\ansicpg1252\deff0\deflang1046{\fonttbl{\f0\fnil\fcharset0 Times
New Roman;}}\viewkind4\uc1\pard\ul\b\f0\fs20 RTF\ulnone\b0\par}
```

FIGURA 2.1A – DOCUMENTO FORMATADO EM RTF

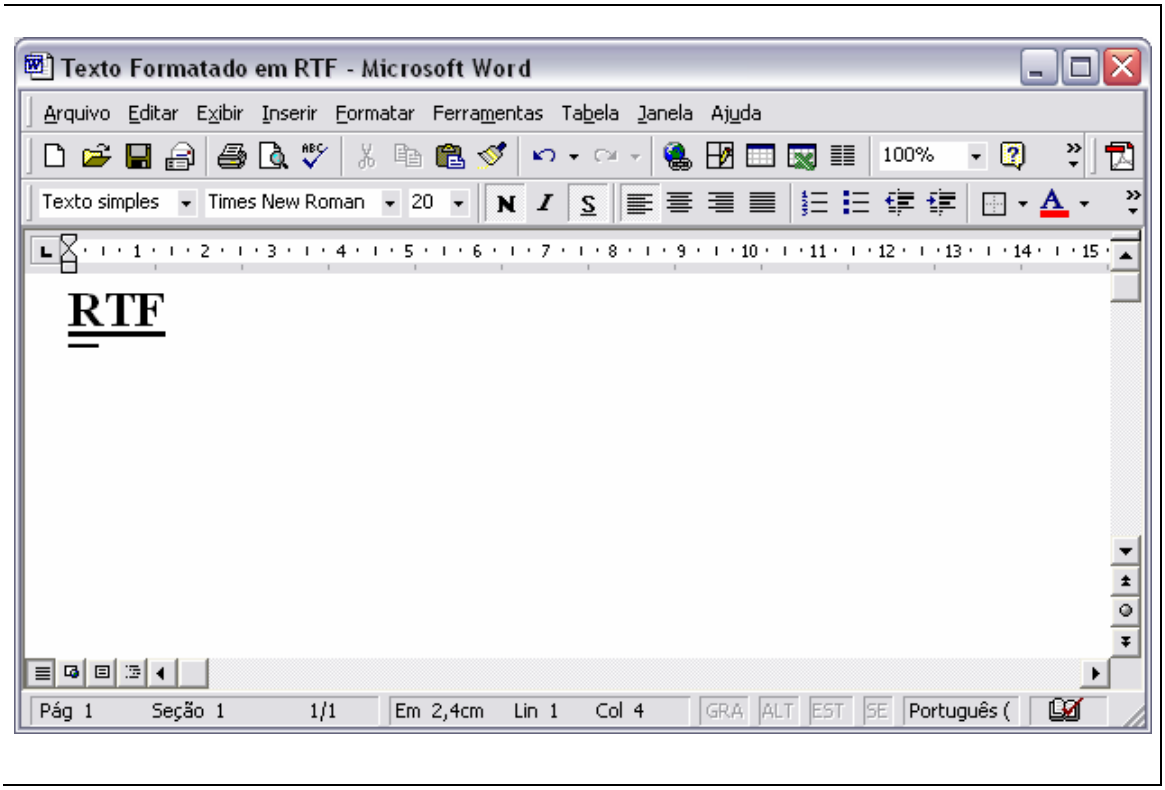

FIGURA 2.1B – APRESENTAÇÃO DE DOCUMENTO RTF NO MS-WORD2000

A abordagem de marcação procedimental adotada pelo formato RTF enseja três grandes problemas [MARCHAL 00]:

- Ausência da estrutura do documento. Percebemos que o usuário deduz a aparência do documento a partir da estrutura do mesmo, mas apenas o resultado do processo é gravado. As informações que levaram esse usuário a um dado conjunto de marcação não está disponível;

- Inflexibilidade. Qualquer mudança nas regras de formatação implicará numa alteração manual do documento. Além disso, a marcação é em maior

ou menor grau dependente do sistema, baseando-se em fontes ou dispositivos de saída particulares, reduzindo a portabilidade;

- Lentidão no processo. Por sua própria natureza o processo é lento e propenso a erros, sendo relativamente fácil formatar incorretamente um documento.

2.1.2 Codificação Genérica

A codificação genérica pode ser vista como marcação envolvida por codificação com a introdução de macros. Tais macros substituem os controles por chamadas de procedimentos externos de formatação. Um identificador genérico (*Generic Identifier* - GI) ou *tag* é anexado a cada elemento de texto e, além disso, regras de formatação são associadas a cada *tag*. Um módulo de formatação processa o texto e produz um documento no formato do dispositivo de saída.

Para efeito de ilustração podemos citar o T_eX. [MARCHAL 00]. O T_eX consiste de um compilador[7] que processa um arquivo com extensão .TEX contendo o documento escrito em algum formato T_eX válido (os formatos mais usados são o *Plain TeX* e *Látex*). O processo de compilação gera um arquivo com extensão DVI, contendo o documento pronto para ser visualizado na tela do computador ou impresso. Necessita-se um editor de texto para escrever o arquivo T_eX. Estão disponíveis vários editores que se acomodam muito bem na edição de documentos T_eX, como por exemplo TeXnic Center, TeXshell, WinShell, Winedt, (X)Emacs, TeXtures [KOPKA 99].

Os benefícios da codificação genérica sobre a marcação procedimental podem ser sumarizados em duas características principais [MARCHAL 00]:

- A portabilidade e flexibilidade são aumentadas. Para alterar a aparência de um documento basta adaptar as macros a ele associadas. A mudança das macros reflete-se automaticamente no documento, não sendo necessário um trabalho manual de recodificação da marcação aplicada no texto;

[7] compilador pode ser grosseiramente definido como um programa que processa um arquivo texto escrito seguindo regras gramaticais de uma linguagem específica, gerando um “arquivo-resultado”

- A marcação descreve a estrutura.

2.2 SGML

Em 1986 foi publicado pela ISO (*International Standardization for Organization*) o padrão ISO 8879 [ISO 86], definindo a SGML (*Standard Generalized Markup Language*), um sistema de marcação generalizada, baseado nos trabalhos feitos pelo Dr. Charles Goldfarb, da IBM. A SGML oferece um esquema de marcação simples, independente de plataforma e extremamente flexível, ela é simplesmente uma maneira de representar documentos [IBM 01].

Analogamente, SGML aproxima-se da codificação genérica, porém acrescida de duas características [MARCHAL 00]:

- a marcação descreve a estrutura do documento, não sua aparência e
- a marcação é coerente com um modelo, semelhante a um esquema de banco de dados. Sendo assim, a marcação pode ser processada por um software ou armazenada em uma base de dados.

A abordagem da SGML não é impor um conjunto próprio de *tags*, mas propor uma linguagem para autores descreverem a estrutura de seus documentos e formatá-los de forma coerente [MARCHAL 00].

Como já fora comentado no primeiro parágrafo dessa seção, a grande diferença entre a codificação genérica e a SGML é o fato de que nesta última, a marcação descreve a estrutura do documento. Assim, a SGML não é uma linguagem, mas uma ferramenta para definir linguagens [NAKHIMOVSKY 00]. Nesta característica, muito próxima a muitas linguagens de programação, encontra-se o poder da SGML, tornando-a extremamente flexível e aberta para novas aplicações.

A estrutura do documento é escrita em um DTD (*Document Type Definition)*, algumas vezes referenciado como aplicação SGML. Um DTD especifica os elementos, seus relacionamento e o conjunto *tags* que efetuam a marcação do documento. Este

outro diferencial da SGML com relação à codificação genérica: na SGML a marcação obedece a um modelo.

Apesar de extremamente poderosa e flexível, a SGML é muito genérica, extensa e complexa: somente o documento de especificação da linguagem possui 500 páginas acrescidas de mais 100 páginas de anexo [HOLZNER 01]. Aplicações reais passaram a exigir derivações da SGML mais reduzidas e mais facilmente implementáveis.

2.3 HTML

Certamente a aplicação mais conhecida, e efetivamente utilizada, da SGML é a HTML (*Hyper Text Markup Language*). Na WWW (*World Wide Web)*, a linguagem de marcação HTML é amplamente empregada para descrever como os *browsers Web*[8] devem exibir os documentos, criada por Tim Berners Lee e Robert Caillau no CERN. Formalmente, HTML é uma aplicação da SGML, isto é, um conjunto de *tags* que seguem as regras da SGML. O conjunto de *tags* definidos pela HTML é adaptado para a estrutura de documentos hipertexto.

A fig. 2.2a apresenta um documento codificado em HTML. Na fig. 2.2b é mostra sua respectiva exibição no *browser Web* MS-Internet Explorer 6.

```
<html>
<head>
<title>Texto Formatado em HTML</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF">
<font face="Times New Roman, Times, serif"><u><b>HTML </b></u></font>
</body>
</html>
```

FIGURA 2.2A – DOCUMENTO CODIFICADO EM HTML

[8] Uma aplicação cliente que permite ao usuário visualizar documentos HTML contidos na *World Wide Web*, em outra rede ou no computador do usuário, acompanhar os vínculos de hipertexto e transferir arquivos.

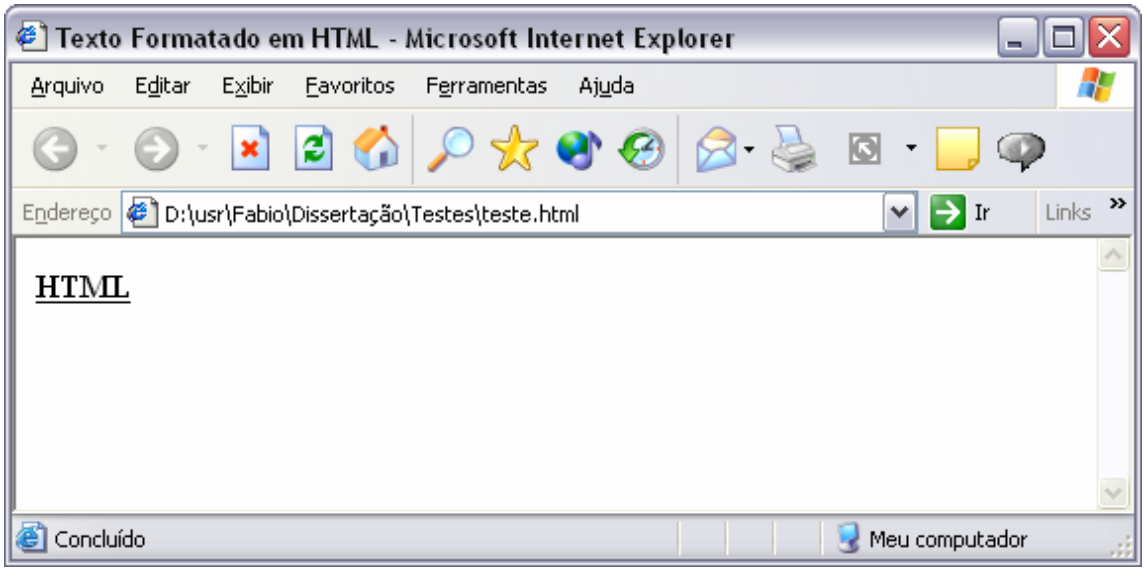

FIGURA 2.2B – APRESENTAÇÃO DE DOCUMENTO NO MS-INTERNET EXPLORER 6

A HTML não encoraja nenhum tipo de estrutura rígida. Na verdade, força uma estruturação muito fraca.

Historicamente, a HTML foi desenvolvida em duas direções contraditórias. Primeiramente, muitas *tags* de marcação foram introduzidas, mas atualmente HTML é de certa forma uma linguagem de marcação procedimental. *Tags* nesta categoria incluem <CENTER> e <FONT>. Tais *tags* são utilizadas para expressar a apresentação, não somente a estrutura. Ao mesmo tempo, atributos de classe e folhas de estilo (*style sheets*) foram adicionados ao HTML, tornando-a parcialmente uma linguagem de codificação genérica. Assim, a HTML, do ponto de vista semântico da marcação, é uma linguagem impura, reunindo componentes apenas de marcação (parágrafos, cabeçalhos, divisões e outros) mesclados a componentes que especificam a aparência do documento (cor do texto, tipo de fonte e outros) [GRAHAM 00].

Ao longo dos anos, para acomodar sua popularidade crescente, a HTML foi estendida várias vezes. A primeira versão possuía pouco mais de uma dezena de *tags*, ao passo que a versão atual possui mais de 100 *tags*, não considerando as *tags* específicas dos diferentes *browsers Web*. Além disso, muitas tecnologias, desenvolvidas pelo W3C

(*World Wide Web Consortium*)[9] ou pela própria indústria, foram introduzidas sendo exemplos notáveis JavaScript[10], Java, Flash, CGI (*Common Gateway Interface*)[11], ASP[12] (*Active Server Pages*), *streamming*[13] de vídeo, MP3[14] dentre outras. O W3C é um grupo responsável pela normatização da *Web*, composto por mais de 400 organizações membro interessadas na *Web* está baseado no MIT (*Massachussets Institut Of Technology*), no INRIA (*Instituit National de Recherche em Informatique et em Automatique*) e na Universidade de Keio, Campus Shonan Fujisawa, Japão.

Essas adições tornaram HTML uma linguagem complexa. A combinação de *tags* é quase infinita e o resultado de uma combinação particular pode ser diferente de um *browser* para outro. Como agravante, desponta a necessidade da criação de novas tags para atender às especificidades das aplicações emergentes como comércio eletrônico (*electronic commerce*) e *streaming* de áudio e vídeo [MARCHAL 00].

Um outro problema, também relacionado, é o emprego excessivo de *tags* para formatar um documento HTML. Não é incomum encontrar-se documentos que possuem significativamente mais marcação que conteúdo, acarretando em um acréscimo de tempo para as operações de *download* e exibição de tais documentos [MARCHAL 00].

Essas limitações já citadas e outras mais vêm se evidenciando, levando alguns a considerar um momento de colapso para a HTML. Tal problemática, somada a grande inserção e aceitação da linguagem HTML como meio de publicação de documentos na *Web*, motivou o W3C a desenvolver XML.

2.4 XML

[9] Todas as especificações do W3C estão publicadas em HTML (e, recentemente XHTML) no site *http://www.w3c.org*

[10] Linguagem de criação de scripts desenvolvida pela *Netscape Communications* e pela *Sun Microsystems, Inc*., que é ligeiramente relacionada à Java.

[11] A especificação que define a comunicação entre servidores de comunicação (como servidores HTTP) e recursos no computador *host* do servidor, como bancos de dados e outros programas

[12] uma linguagem para geração de páginas HTML dinamicamente através de um servidor

[13] é um mecanismo para enviar conteúdo multimídia (vídeo, áudio, animações) através de uma rede corporativa ou Internet., onde tais conteúdos podem ser visualizados pelo cliente na medida em que são recebidos

[14] MPEG-1 Layer 3, um padrão internacional para alta compressão de arquivos de sons sem perda significativa de qualidade

2.4.1 Origens

Jon Bosak, engenheiro da Sun Microsystems e grande conhecedor da SGML, após ter transferido para um sistema on-line as 150 mil páginas de documentação do Novell Netware, da Novell, utilizando exclusivamente a SGML, propôs ao W3C, o estudo de uma forma de aproveitar o poder da SGML para aplicações voltadas à *Web*.

Dos trabalhos de Bosak, Tim Bray e Sperberg McQueen especialistas em SGML na Sun MicroSystems, em 1996 foi definido um sistema de codificação denominado XML (*Extensible Markup Language*) baseado no SGML para permitir a distribuição de informação heterogênea através da *Web*. Muitos dos recursos SGML tiveram de ser abolidos para que a XML fosse leve e pequena o suficiente para se tornar eficaz e menos complexa. Em fevereiro de 1998, a XML tornou-se uma especificação formal, reconhecida pelo W3C.

2.4.2 Definição de XML

XML não é em si mesma uma linguagem, mas um conjunto de regras para projetar linguagens de marcação [RAY 01]. Tecnicamente, XML é considerada uma metalinguagem baseada em um subconjunto da SGML [HOLZNER 01]. Metalinguagem é uma linguagem que permite a especificação de outra linguagem [SEBESTA 00]. Neste texto, XML será referenciada com o termo *linguagem*. A XML oferece o panorama de uma ampla variedade de aplicações, cada uma servindo a uma função em particular e usando a WEB como uma infra-estrutura de distribuição.

Segundo [MCGRATH 99], XML surgiu como resultado de uma longa e cuidadosa análise, do real significado do termo "documento" no meio digital, que de uma forma geral, é composto de três elementos distintos:

- conteúdo de dados – as palavras propriamente ditas;
- estrutura – corresponde ao tipo de documento e a organização de seus elementos, a ordem com que eles aparecem; e finalmente

- apresentação – a forma com que as informações são mostradas ao leitor, ou seja, na tela através de um *browser Web*, por voz, em um aparelho de telefone celular, em um computador portátil, entre outros.

Conforme discutido na seção 2.3, HTML é uma aplicação da SGML com um objetivo particular: exibir documentos hipertexto na *Web*. XML, por sua vez, é um perfil da SGML e pode dar suporte a uma infinidade de outras aplicações diferentes da HTML. Para o W3C [W3C 01b] XML define "um dialeto de SGML extremamente simples, o qual é completamente descrito na Especificação XML. O objetivo é habilitar o SGML padrão para que ele possa ser distribuído, recebido e processado na *Web*, da mesma forma que hoje isso é possível com o HTML". Por essa razão, XML foi projetado para apresentar facilidade de implementação e interoperabilidade com ambos, SGML e HTML.

A grande revolução da XML está na sua característica principal que é a dissociação entre a estrutura da apresentação e o conteúdo do documento, permitindo, assim, que um mesmo documento possa ser apresentado através de várias formas: um monitor de um computador pessoal, uma pequena tela de um telefone celular ou até mesmo que ela seja transformada em voz para utilização dos deficientes visuais.

2.4.3 Facilidades XML

Desde a publicação da linguagem XML, seu emprego vem se expandindo de forma bastante vigorosa. Nesta seção são apresentadas algumas facilidades oferecidas pela XML:

- Distinção entre interface, processos e dados. Um dos grandes atrativos do XML é a separação mantida entre dados e interface com o usuário. Por exemplo, HTML especifica como os dados serão exibidos no *browser* [GRAHAM 00]; XML, por sua vez, simplesmente define o conteúdo, valendo-se de delimitadores simplesmente para descrição de dados (índices econômicos, dados bibliográficos). Para definir o modo de apresentação em *browsers* de dados descritos em XML, existem

stylesheets como a XSL (*Extensible Style Language*) e a CSS (*Cascading Style Sheets*) - desta forma, o desenvolvedor tem a liberdade de definir a interface e o processamento dos dados, bastando utilizar diferentes *stylesheets* e aplicações. Tal separação entre dados e apresentação traz consigo uma nova funcionalidade - a integração de dados oriundos de diferentes fontes. As diferentes informações de interesse podem ser convertidas para o formato XML em uma camada intermediária (por exemplo, um servidor de aplicações), permitindo um intercâmbio de dados tão simples quanto à exibição de páginas HTML. Esses dados, já codificados em XML, podem ser distribuídos diretamente à máquina cliente, utilizando o formato HTTP[15] (já suportado) e sem a necessidade de novas requisições aos dados legados da estação servidora;

- Flexibilização no intercâmbio de dados: Em [HOLZNER 01], os formatos de dados patenteados ou proprietários são apontados como empecilhos para a troca de dados entre organizações ou mesmo aplicações distintas. Em XML, tanto os dados, como suas respectivas marcações, são armazenados em formato texto, provendo fácil acesso. Linguagens de marcação padronizadas permitem que documentos sejam utilizados para diferentes propósitos;

- Personalização das linguagens de marcação: XML possibilita a criação de linguagens de marcação específicas [FURGERI 01]. Diversas linguagens personalizadas estão sendo padronizadas tais como a CBL (*Common Buiness Library*), a IFX (*Financial Exchange*), a BITS (*Banking Industry Technology Secretariat*), a TIM (*Telecommunications Interchange Markup*), a CML (*Chemical Markup Language*) e muitas outras.

- Autodescrição de dados: Documentos XML, são, grosso modo, autodocumentados. As *tags*, em geral, são mnemônicas e explicitam em si mesmas o tipo de informação que está sendo relacionada. A grande

[15] *Hypertext Transfer Protocol* (HTTP) é um procolo que executa em nível de aplicação para distribuição de documentos hipermída em sistemas de informação. A característica do HTTP de negociar a representação dos dados a serem transmitidos possibilita a independência dos sistemas com relação aos dados a serem transferidos

revolução advinda do uso da XML é o uso da marcação para definir o que é a informação e não como ele deve ser mostrada [BOSAK 99].

- Estruturação e integração dos dados: XML possui uma ênfase forte na exatidão dos documentos. XML impõe um rigor na codificação documentos, evitando práticas desleixadas. Um *browser* XML pode verificar se um documento XML é um documento bem formado (*well-formed document*) e se o mesmo é válido. Os conceitos de boa formação e validade de um documento XML serão apresentados na seção 2.4.5 e 2.4.6, respectivamente.

2.4.4 Estrutura de Documentos XML

Um documento XML é um documento que possui uma estrutura hierárquica com duas seções: Prólogo (*Prolog*) e Corpo (*Body*). O documento XML inicia com um Prólogo opcional, constituído de declarações XML, PIs (*Processing Instructions*), DTDs e comentários. A fig. 2.3 apresenta um documento XML bastante simples.

Elementos dividem o documento em suas partes constituintes. Tais elementos podem conter texto, outros elementos, ou ambos. No documento da fig. 2.3, *Agency* é o elemento raiz, com *BankAccount*, *Number*, *Name*, *Type*, *OpenDate* e *Balance*, sendo seus elementos filho. Elementos são delimitados por uma *tag* de início (ex: <Type>) e uma *tag* de fim (</Type>). Em HTML as *tags* de fim não são necessárias, devido ao tratamento especial implementado pelos *browsers*, mas XML é rígida e exige que todas as *tags* sejam apropriadamente fechadas. As *tags* também devem ser aninhadas corretamente. A seqüência <A><B><C></C></B></A> é válida, mas <A><B><C></B></C></A> não o é. Sobreposição de *tags* não é permitida. Finalmente, documentos XML são *case-sensitive*, isto é, existe distinção entre caracteres maiúsculos e minúsculos.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type=”text/css” href=”estilo.css”?>
<!-- Definição da DTD do documento-->
<!DOCTYPE Agency [
```

```
      <!ELEMENT Agency (BankAccount+)>
      <!ELEMENT BankAccount (Number, Name, Type, OpenDate, Balance)>
      <!ELEMENT Number (#PCDATA)>
      <!ELEMENT Name (#PCDATA)>
      <!ELEMENT OpenDate (#PCDATA)>
      <!ELEMENT Type (#PCDATA)>
      <!ELEMENT Balance (#PCDATA)>
]>
<!--Exemplo de um documento XML -->
<Agency>
      <BankAccount>
            <Number>721174</Number>
            <Name>Carlos Westphall</Name>
            <Type>Pagamento</Type>
            <OpenDate>25/07/1986</OpenDate>
            <Balance>25382.20</Balance>
      </BankAccount>
      <BankAccount>
            <Number>1709459</Number>
            <Name>Fabio Spanhol</Name>
            <Type>Corrente</Type>
            <OpenDate>30/08/1993</OpenDate>
            <Balance>1432.62</Balance>
      </BankAccount>
</Agency>
```

FIGURA 2.3 – UM DOCUMENTO XML SIMPLES

Elementos também podem possuir atributos, os quais são, simplificando, pares de nome e valor em *tags* iniciais, como por exemplo, *<Student RollNo="2">*, onde *RollNo* é o nome do atributo e “2” seu valor associado. XML requer que todos os atributos estejam envolvidos por caracteres ‘ ou “. Atributos são usados, principalmente, com o propósito de modificar o comportamento de um elemento e não para armazenar dados [RAY 01].

Existem atualmente cinco caracteres que possuem semântica especial. Tais caracteres são <, >, ', ", e &. Caso seja necessário utilizar algum destes caracteres diretamente como conteúdo do documento, e não como marcação, deve-se utilizar os códigos de escape *<*, *>*, *'*,*"*, e *&amp*; respectivamente.

Na fig. 2.4 estão destacadas as seções XML (prólogo e corpo) presentes no documento apresentado na fig. 2.3.

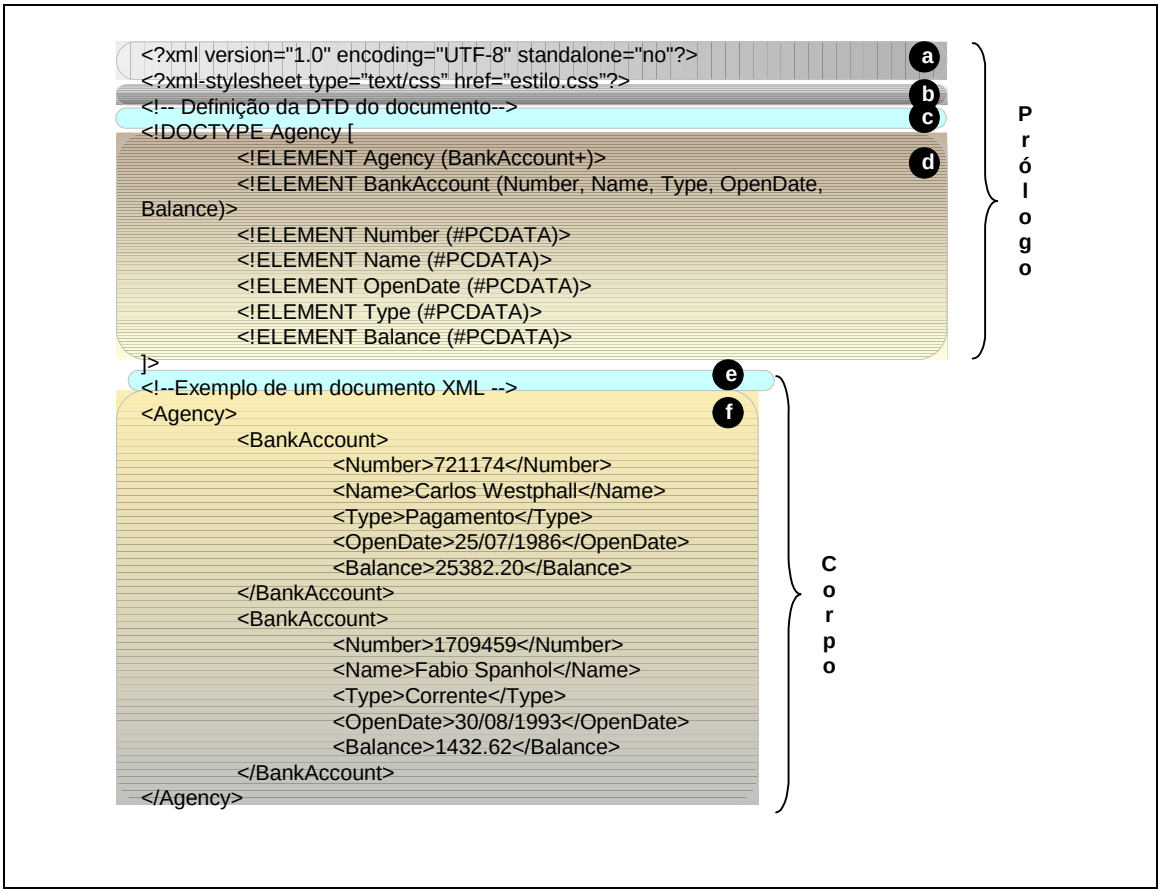

FIGURA 2.4 – ESTRUTURA DE UM DOCUMENTO XML

Caso um documento XML possua a declaração *xml* (fig. 2.4a), ela deve ser a primeira linha do documento, não sendo precedida por nenhum espaço em branco. Essa declaração identifica o documento como sendo um documento XML e possui três atributos: *version* (obrigatório), *encoding* e *standalone* (ambos opcionais).

Documentos XML são documentos texto (*text*). O termo "texto" aqui não se refere ao ASCII (*American Standard Code for Information Interchange*), mas sim ao Unicode. O uso do Unicode permite que os documentos XML sejam produzidos em diferentes idiomas, com os mais variados conjuntos de caracteres e símbolos. Os caracteres reais de um documento são armazenados na verdade utilizando códigos numéricos e o W3C não utiliza o ASCII como conjunto de caracteres padrão para XML, mas sim o Unicode. Essa escolha deve-se a limitação do conjunto ASCII, que utiliza um byte, permitindo

somente de 0 a 255 códigos de caractere, quantidade insuficiente para representar todos os caracteres e ideogramas presentes em documentos tibetanos, cirílicos, arábicos, hebraicos, tailandeses e outros. O Unicode, por sua vez, utiliza 2 bytes, permitindo de 0 a 65.535 códigos de caractere, independentes de plataforma ou linguagem [UC 01], estando atualmente cerca de 40.000 códigos já reservados [HOLZNER 01].

O atributo *encoding* na declaração XML especifica qual codificação está sendo utilizada para representar os caracteres no documento, por padrão os processadores XML assumem que a codificação utilizada é a UTF-8.

Finalmente, o atributo *standalone* pode possuir os valores *"yes"*, indicando ao processador se o documento XML depende de referências externas, ou *"no"* , quando documento não possui referencias externas. Na fig. 2.4a, a declaração *xml* define o atributo obrigatório *version* com o valor 1.0 (a versão atualmente disponível de XML).

As instruções de processamento (fig. 2.4b) especificamente parâmetros adicionais para o processador XML, sendo iniciadas por *<?* e finalizadas por *?>*. Como o suporte às PI é dependente do processador XML, o conteúdo das PI não está especificado na recomendação XML do W3C [HOLZNER 01].

Os comentários em XML (Fig. 2.4c e 2.4e) assemelham-se aos comentários em HTML, provendo uma descrição útil para manutenção de documentos complexos e uma eficiente documentação *inline*. São iniciados por *<!*—e finalizados com --> [HOLZNER 01].

A última parte apresentada no prólogo indicado na fig. 2.4 é a declaração de tipo de documento. É onde se especifica uma série de parâmetros tais como: declarações de entidade, a DTD que será usada para validar o documento, e o nome do elemento raiz. A DTD (fig. 2.4d), permite ao processador XML comparar a instância do documento com um modelo de documento, um processo chamado *validity checking*. A checagem de validade do documento é opcional, mas quando aplicada, garante que o documento segue padrões pré-determinados e inclui dados requeridos [RAY 01]. As DTD serão discutidas mais detalhadamente na seção 2.4.6.

Como descrito anteriormente, um documento XML possui uma estrutura hierárquica e o diagrama de árvore é uma forma bastante adequada para representar o relacionamento entre as partes do documento em questão. A fig. 2.5 apresenta o documento da fig. 2.3 em um diagrama de árvore. Cada um dos retângulos da fig. 2.5 é um nodo da árvore. Ainda na fig. 2.5, os retângulos pretos representam os elementos, enquanto os retângulos brancos, denominados folhas, representam o conteúdo real do documento. Documentos XML possuem um, e somente um, elemento raiz (*root element*), também denominado elemento documento (*document element*), pois ele inclui todos os outros elementos, definindo assim os limites do documento [RAY 01]. Todos os outros elementos são filhos do elemento raiz. O elemento raiz da árvore da fig. 2.5 está identificado pelo rótulo *Agency*.

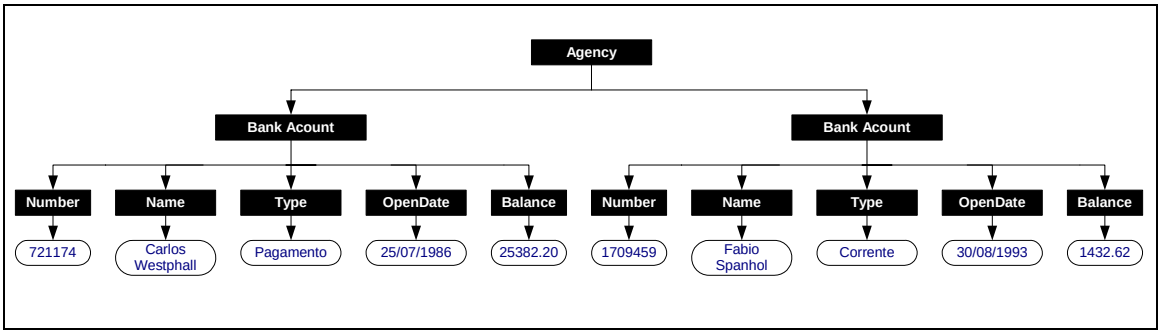

FIGURA 2.5 – DIAGRAMA EM ÁRVORE DE UM DOCUMENTO XML

2.4.4 Documentos XML bem formados

Todo documento XML possui uma estrutura lógica e física. Fisicamente, o documento é composto por unidades denominadas entidades. Entidades são composições formadas por seqüências de caracteres (texto) ou padrões binários, mas não ambos. O termo entidade define a unidade de armazenamento de dados em XML [HOLZNER 01].

Uma entidade pode referenciar-se a outra entidade para efetuar a inclusão desta no documento. Um documento inicia com um elemento raiz (*root*) ou entidade *document*. Logicamente, o documento é composto por declarações, elementos, comentários, referências de caractere, e instruções de processamento, todos os quais são indicados no

documento por uma marcação explícita. As estruturas lógicas e físicas devem estar corretamente aninhadas [W3C 01b].

Informalmente, um documento bem formado é um documento que contém um ou mais elementos, e o elemento raiz, contém todos os elementos restantes [HOLZNER 01]. Formalmente, o W3C define que um objeto textual é bem formado se:

- tomado como um todo, ele corresponde à produção intitulada *document;*
- tal documento atende a todas as restrições de boa formação dadas na especificação XML [W3C 01b];
- Cada uma das entidades analisadas, referenciada direta ou indiretamente dentro do documento, é bem formada;

2.4.5 Documentos XML Válidos

Um documento XML é válido se possuir uma DTD associada a ele, além de obedecer às regras expressas na declaração [W3C 01b].

A sintaxe correta de um documento é especificada na DTD, que pode estar definida no próprio documento ou armazenada em arquivos separados. Formalmente a DTD define a gramática para uma classe de documentos.

Existem ferramentas chamadas *validadores* que permitem verificar se um documento XML é bem formado e válido. Alguns exemplos são o W3CXML Validator[16], STG XML Validation[17] e o *validador*[18] do Language Technology Group, da Universidade de Edinburgh.

2.4.6 Esquemas XML

[16] Disponível em *validator.w3.org/*
[17] Disponível em *www.stg.brown.edu/service/xmlvalid/*
[18] Disponível em *www.ltg.ed.uk/~richard/xml-check.html*

Existe um conflito potencial entre facilidade de uso e flexibilidade. De maneira geral, soluções mais flexíveis são mais trabalhosas e soluções específicas são otimizadas para certas tarefas. Construindo um paralelo entre HTML e XML destaca-se esse fato. Ambas são utilizadas para publicar documentos na *Web* (apesar de XML aplicar-se a uma série de outros propósitos). HTML possui um conjunto fixo de *tags* e os softwares que a manipulam podem ser otimizados, como, por exemplo, os editores HTML que oferecem *templates*, edição visual, *preview* do documento e outras facilidades [MARCHAL 00]. XML, por outro lado é uma solução flexível. Ele não predefine *tags*, mas auxilia na estruturação dos documentos. Logo, editores XML devem aceitar qualquer tipo de estrutura de documento e os documentos XML obedecem a uma sintaxe definida pelo criador do documento. Mas, uma vez definida uma estrutura, o documento deve obedecê-la para ser válido.

Um documento XML para ser considerado válido deve obedecer a um conjunto de regras sintáticas denominado esquema [HOLZNER 01]. O esquema descreve um modelo para uma classe de documentos [VLIST 01]. No processo de validação é realizada a comparação de um documento XML com seu esquema correspondente, sendo um documento XML considerado válido caso sua sintaxe tenha sido comparada com sucesso na validação [CASTRO 01]. Assim, os esquemas especificam a estrutura e a sintaxe do documento XML e não seu conteúdo.

Um esquema também pode ser visto como um acordo sobre um vocabulário comum para uma aplicação particular que envolve a troca de documentos [VLIST 01].

Esquemas podem ser escritos em dois sistemas distintos o XML *Schema*, padrão recente desenvolvido pelo W3C; e DTD (*Document Type Definition*), um sistema de regras antiquado, embora amplamente utilizado.

Muitos dos analisadores XML, como o embutido no *browser* MS-Internet Explorer, exigem que um documento XML seja bem formado, mas não necessariamente válido. Contudo, caso um esquema esteja disponível ele será utilizado para validar o documento.

2.4.6.1 DTD

A DTD é um sistema para definição de esquemas antigo, mas ainda amplamente utilizado. Baseado em regras, possui uma sintaxe peculiar, embora bastante limitada [CASTRO 01].

DTD apresenta uma série de desvantagens se comparada a XML *Schema*. Primeiramente, a própria sintaxe DTD possui pouca relação com XML e não pode ser validada por um *parser* XML. Além disso, as declarações DTD são globais, impedindo a existência de nomes idênticos, mesmo em contextos distintos. Finalmente, DTDs não podem controlar o tipo de informação que um determinando elemento ou atributo do documento XML pode conter [CASTRO 01].

2.4.6.1.1 Declarações de Elemento

A DTD é um mecanismo para descrever todo objeto (elemento, atributo e outros) que pode aparecer no documento. Para declarar um elemento numa DTD utilizam-se regras que obedecem a seguinte sintaxe: *<!ELEMENT NOME MODELO_CONTEÚDO>*. O lado direito dessa regra *(MODELO_CONTEÚDO)* define o lado esquerdo (*NOME*). Em outras palavras, o modelo de conteúdo (*MODELO_CONTEÚDO*) lista todas derivações aceitáveis no elemento.

Para a maioria dos elementos, *MODELO_CONTEÚDO* é uma lista de elementos. Também podem ser empregas as seguintes palavras-chave:

- *#PCDATA* (*Parsed Character Data*) indica que o elemento deve armazenar apenas texto puro, não contendo marcação. *#PCDATA* é freqüentemente, mas não apenas, empregado em elementos folha. Elementos folha são elementos que não possuem elementos filho.

- *EMPTY* representa um elemento vazio, sempre indicando um elemento folha.

- *ANY* indica que um elemento pode conter outro elemento declarado na DTD. É raramente utilizada, pois carrega pouca informação sobre a

estrutura. ANY é utilizada, em geral, durante o desenvolvimento da DTD, antes que uma regra precisa tenha sido escrita. Os elementos devem estar declarados na DTD.

O modelo de conteúdo que aceita tanto *#PCDATA* quanto outros elementos é denominado *modelo de conteúdo misto*. Já modelos de conteúdo que contêm apenas elementos são denominados *modelo de conteúdo de elemento.*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE memo [
<!ELEMENT memo (de, para, cc, data, conteudo) >
<!ELEMENT de (#PCDATA) >
<!ELEMENT para (#PCDATA) >
<!ELEMENT cc (#PCDATA) >
<!ELEMENT data (#PCDATA) >
<!ELEMENT conteudo (p+) >
<!ELEMENT p (#PCDATA) >
]>
<memo>
<de><Francisco</de>
<para>João da Silva</para>
<cc>Luis Alberto</cc>
<data>18 de novembro de 1999.</data>
<conteudo><p>Este exemplo mostra a declaração da DTD e dos elementos que
compõem a estrutura.</p></conteudo>
<p>A XML é muito rigorosa quanto as endtags, deve-se ter o cuidado de não
esquece-las</p>
</conteudo>
</memo>
```

Como citado anteriormente, um elemento pode conter vários filhos. O W3C definiu para as DTDs uma sintaxe bastante próxima das *expressões regulares* para definir como se dará esse relacionamento. Essa sintaxe é composta de símbolos denominados indicadores de ocorrência e conectores [MARCHAL 00], [HOLZNER 01].

Os indicadores de ocorrência especificam a quantidade de vezes que um elemento filho pode aparecer no documento. Os indicadores são o asterisco (*), o sinal de adição (+) e o sinal de interrogação (?). A semântica dos indicadores é a seguinte:

- um elemento que não seja seguido por um indicador de ocorrência deve ocorrer uma, e somente uma, vez no elemento definido;
- um elemento seguido por um indicador + deve ocorrer uma ou mais vezes no elemento definido. O elemento pode repetir-se;
- um elemento seguido por um indicador * pode ocorrer nenhuma ou várias vezes no elemento definido. O elemento é opcional, e se incluído, pode repetir-se indefinidamente;
- um elemento seguido por um indicador ? pode ocorrer nenhuma ou apenas uma vez no elemento definido. O elemento é opcional, e se incluído, não pode repetir-se.

Na fig. 2.6 é mostrado o uso de indicadores de ocorrência. Os elementos-filho permitidos para o elemento *entry* são *name*, *address*, *tel*, *faz* e *email*. Com exceção de *name*, os outros filhos são opcionais e podem repetir-se, pois foram sufixados por um indicador *. Já para o elemento *address* os elementos-filho possíveis são *street*, *region*, *postal-code*, *locality* e *country*. Nenhum desses elementos-filho pode repetir, mas *region* é opcional.

```
<!ELEMENT entry (name,address*,tel*,fax*,email*)>
<!ELEMENT address (street,region?,postal-code,locality,country)>
```

FIGURA 2.6 – EMPREGO DE INDICADORES DE OCORRÊNCIA DTD

Os conectores determinam a ordem na qual os elementos filhos podem aparecer no documento. Os conectores são os caracteres barra vertical ou *pipe* (|) e a vírgula (,). A semântica dos conectores é a seguinte:

- O caractere “,” indica que ambos os elementos, do lado esquerdo e direito da vírgula, devem aparecer na mesma ordem no documento.
- O caractere “|” indica que somente um dos elementos, do lado esquerdo ou direito da barra vertical deve estar presente no documento.

A fig. 2.7 apresenta um exemplo do uso de conectores, definindo os elementos *name* e *address*. Observando o fragmento de código da fig. 2.7 e a semântica dos conectores apresentada anteriormente, nota-se que elementos-filho possíveis para o elemento *name* são *#PCDATA* ou *first_name* ou *last_name*, isto é, apenas um dos elementos listados. Todo o modelo pode-se repetir pelo emprego do indicador de ocorrência *. Ainda com relação a fig. 2.7, elementos-filho aceitáveis para *address* são *street*, *region*, *postal-code*, *locality* e *country*, nesta ordem exata.

```
<!ELEMENT name (#PCDATA | first_name | last_name)*>
<!ELEMENT address (street,region?,postal-code,locality,country)>
```

FIGURA 2.7 – EMPREGO DE CONECTORES DTD

É válido ressaltar que os vários componentes de um *modelo de conteúdo misto* devem sempre estar separados por uma "|" e o modelo deve repetir [MARCHAL 00]. Assim, uma regra da forma *<!ELEMENT name (#PCDATA, fname, lname)>* está incorreta e deve ser substituída por *<!ELEMENT name (#PCDATA | fname | lname)*>*.

Caso as DTDs tornem-se extensas e complexas, a especificação XML permite a inclusão de comentários dentro das DTDs. Os comentários na DTD são os mesmos utilizados nos documentos XML, isto é, a marcação *<!—comentário -->*.

2.4.6.1.2 Declaração de Atributos

Como já fora discutido, os atributos (*Attributes*) são pares especiais nome e valor que podem ser empregados em *tags* de início e *tags* vazias para oferecer informações adicionais sobre um elemento. O uso de atributos permite um detalhamento mais claro na descrição dos elementos que compõem o documento, possibilitando definir nessa descrição algum aspecto do comportamento do elemento ou criar um subtipo [RAY 01].

Para que um documento XML que utilize atributos seja válido, tais atributos também devem ser declarados em uma DTD. Elementos atributo são declarados com a cláusula *ATTLIST*. Por exemplo, a declaração *<!ATTLIST tel preferred (true | false) "false">* especifica que a marcação *ATTLIST* possui um elemento *tel*, um atributo

preferred associado ao tipo *(true|false)* e um valor padrão *"false"*. Para elementos que possui mais de um atributo, as declarações podem ser agrupadas como em *<!ATTLIST email href CDATA #REQUIRED preferred (true | false) "false">* [MARCHAL 00].

Apesar de poderem estar dispostos em qualquer posição na DTD, para facilitar a legibilidade, é aconselhável que os atributos estejam imediatamente após a declaração do elemento correspondente.

A DTD provê um controle mais apurado sobre o conteúdo de atributos que sobre o conteúdo de elementos. De modo geral os atributos são divididos em três categorias:

- String: Contêm texto *<!ATTLIST email href CDATA #REQUIRED>*

2.4.6.1.3 Declaração de Tipo de Documento

A declaração de tipo de documento (*document type declaration)* liga uma DTD ao documento. Apesar de similares, os termos declaração de tipo de documento (*document type declaration*) e a *DTD* não representam os mesmos conceitos [MARCHAL 00], [RAY 01]. A declaração de tipo de documento possui a seguinte forma: *<!DOCTYPE elemento_raiz uri_da_DTD [subconjunto_interno]>*.

2.4.6.1.4 DTD interna e externa

As DTD podem ser internas ou externas. A DTD interna é inserida no próprio documento, enquanto que a DTD externa aponta para uma entidade externa. As DTD externas e internas diferem na definição de parâmetros. A DTD interna é incluída entre colchetes na definição de tipo do documento. Já a DTD externa é armazenada em uma entidade separada e referenciada através da definição de tipo do documento.

2.4.6.2 XML *Schema*

O modelo DTD apresenta uma série de desvantagens se comparada a XML Schema. Primeiramente, a própria sintaxe DTD possui pouca relação com XML e não pode ser validada por um parser XML. Além disso, as declarações DTD são globais, impedindo a existência de nomes idênticos, mesmo em contextos distintos. Finalmente, DTDs não podem controlar o tipo de informação que um determinando elemento ou atributo do documento XML pode conter.

A modelo XML *Schema* foi proposto pelo W3C como alternativa para as limitações das DTDs, tendo sido aprovada como recomendação em maio de 2001. Escrita em XML, XML *Schema* permite tanto elementos globais, como elementos locais. Disponibiliza ainda um sistema de tipos de dados, permitindo especificar qual tipo um determinado elemento deve conter [CASTRO 01]. Muitos dos analisadores XML, como o embutido no browser MS-Internet Explorer, exigem que um documento XML seja bem formado, mas não necessariamente válido. Contudo, caso um esquema esteja disponível ele será utilizado para validar o documento.

2.4.7 Folhas de Estilo em XML

A ênfase na criação de documentos XML é definir uma estrutura para as informações nele contidas. Assim, os documentos não contêm nenhum tipo de informação referente à apresentação das informações, apenas a estrutura e a semântica [HOLZNER 01]. Para que, por exemplo, um *browser* apresente um documento XML corretamente deve ser fornecida uma *folha de estilo* (*stylesheet*), responsável pela aparência que o documento XML deve ter quando exibido [RAY 01], [NAKHIMOVSKY 00]. Basicamente, para a criação de folhas de estilos pode-se utilizar modelos de formatação da CSS[19] (*Cascading Style Sheets*), que é a linguagem de folhas de estilos que surgiu no final de 1996 e é utilizada por HTML; ou a XSL[20] (*Extensible Style Language)*, proposta pelo W3C para suprir a carência de um modelo de apresentação dos documentos XML.

[19] O endereço http://www.w3.org/Style/CSS/ apresenta a programação CSS segundo a visão do W3C
[20] Especificada em http://www.w3.org/Style/XSL/

A fig. 2.8 mostra um documento XML que contém o trecho inicial da obra “Os Sertões”, do escritor brasileiro Euclides da Cunha[21] e na fig. 2.9 é apresentada a visualização do documento no *browser* MS-Internet Explorer 5.5, sem o auxílio de uma folha de estilo.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DOCUMENT>
<TITLE>Os Sert&#x00f5;es</TITLE>
      <AUTHOR>Euclides da Cunha</AUTHOR>
      <SECTION>Nota preliminar</SECTION>
      <P>
            Escrito nos raros intervalos de folga de uma carreira fatigante,
este livro, que a princ&#x00ed;pio se resumia &#x00e0; hist&#x00f3;ria da
Campanha de Canudos, perdeu toda a atualidade, remorada a sua
publica&#x00e7;&#x00e3;o em virtude de causas que temos por escusado apontar.
      </P>
      <P> Demos lhe, por isto, outra fei&#x00e7;&#x00e3;o, tomando apenas
variante de assunto geral o tema, a princ&#x00ed;pio dominante, que o sugeriu.
      </P>
      <P>   Intentamos esbo&#x00e7;ar, palidamente embora, ante o olhar de
futuros historiadores, os tra&#x00e7;os atuais mais expressivos das sub-
ra&#x00e7;as sertanejas do Brasil. E faz&#x00ea;mo-lo porque a sua
instabilidade de complexos de fatores m&#x00fa;ltiplos e diversamente
combinados, aliada &#x00e0;s vicissitudes hist&#x00f3;ricas e
deplor&#x00e1;vel situa&#x00e7;&#x00e3;o mental em que jazem, as tomam talvez
ef&#x00ea;meras, destinadas a pr&#x00f3;ximo desaparecimento ante as
exig&#x00ea;ncias crescentes da civiliza&#x00e7;&#x00e3;o e a
concorr&#x00ea;ncia material intensiva das correntes migrat&#x00f3;rias que
come&#x00e7;am a invadir profundamente a nossa terra.
      </P>
      <P>   O jagun&#x00e7;o destemeroso, o tabar&#x00e9;u ing&#x00ea;nuo e o
caipira simpl&#x00f3;rio ser&#x00e3;o em breve tipos relegados &#x00e0;s
tradi&#x00e7;&#x00f5;es evanescentes, ou extintas.
      </P>
</DOCUMENT>
```

FIGURA 2.8 – DOCUMENTO XML *SERTOES.XML*

[21] Disponível em *http://www.bibvirt.futuro.usp.br/acervo/literatura/autores/euclidesdacunha/sertoes/sertoes.html*

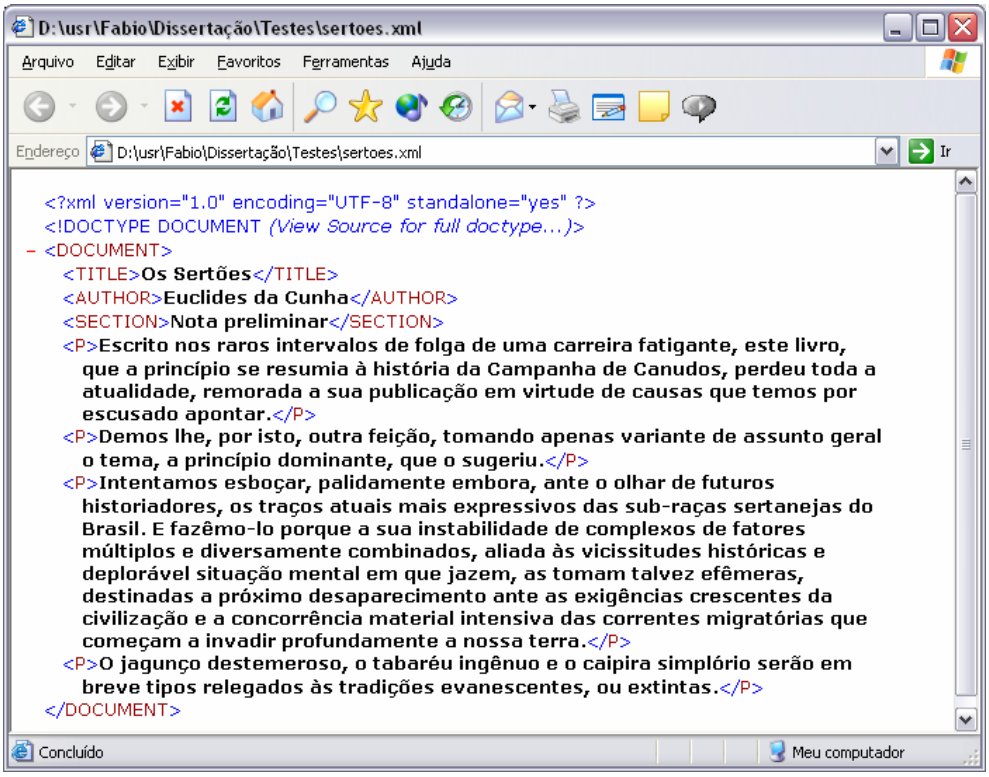

FIGURA 2.9 – *BROWSER* MS-INTERNET EXPLORER APRESENTANDO UM DOCUMENTO XML SEM FOLHA DE ESTILO ASSOCIADA

2.4.7.1 CSS

As folhas de estilo CSS (*Cascading Style Sheets*) são constituídas por *listas de regras de estilo* [HOLZNER 01], sendo tal lista ligada a um documento para indicar como os elementos contidos no documento devem ser apresentados. Uma regra consiste em um seletor, especificando o elemento ou elementos aos quais a regra deve ser aplicada e a regra propriamente dita. Tal regra é delimitada entre chaves, isto é, { e }; sendo composta por propriedades a serem estilizadas e os valores que devem ser aplicados na estilização. Um seletor pode possuir várias propriedades estilizadas numa única linha de comando, bastando para isso separar cada propriedade com um caractere de "ponto-e-vírgula". A fig. 2.10, apresenta *estilo.css*, uma folha de estilo para formatar o documento XML mostrado na fig. 2.8, da seção 2.4.7.

```
TITLE {display: block; font-family: sans-serif; font-size: 34 pt; font-weight:
bold;  Text-align: center; text-decoration: underline; background-color:
#ddffff}
AUTHOR {display: block; font-size: 18 pt; font-weight: bold;
       Text-align: center }
SECTION {display: block; font-size: 16 pt; font-weight: bold;
       Text-align: center; font-style: italic}
P {display: block; margin-top: 10}
```

FIGURA 2.10 – FOLHA DE ESTILO *ESTILO.CSS* PARA O DOCUMENTO *SERTOES.XML*

A folha de estilo (arquivo css) deve ser ligada ao documento XML. Essa ligação é feita através da instrução de processamento *<?xml-stylesheet?>*, setando-se o atributo *type* para *"text/css"* e o atributo *href* como o URI da folha de estilo, conforme destacado na fig. 2.11.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?xml-stylesheet type="text/css" href="estilo.css"?>
<DOCUMENT>
<TITLE>Os Sert&#x00f5;es</TITLE>
       <AUTHOR>Euclides da Cunha</AUTHOR>
       <SECTION>Nota preliminar</SECTION>
       <P>    Escrito nos raros intervalos de folga de uma carreira fatigante,
este livro, que a princ&#x00ed;pio se resumia &#x00e0; hist&#x00f3;ria da
Campanha de Canudos, perdeu toda a atualidade, remorada a sua
publica&#x00e7;&#x00e3;o em virtude de causas que temos por escusado apontar.
       </P>
       <P>    Demos lhe, por isto, outra fei&#x00e7;&#x00e3;o, tomando apenas
variante de assunto geral o tema, a princ&#x00ed;pio dominante, que o sugeriu.
       </P>
       <P>    Intentamos esbo&#x00e7;ar, palidamente embora, ante o olhar de
futuros historiadores, os tra&#x00e7;os atuais mais expressivos das sub-
ra&#x00e7;as sertanejas do Brasil. E faz&#x00ea;mo-lo porque a sua
instabilidade de complexos de fatores m&#x00fa;ltiplos e diversamente
combinados, aliada &#x00e0;s vicissitudes hist&#x00f3;ricas e eplor&#x00e1;vel
situa&#x00e7;&#x00e3;o mental em que jazem, as tomam talvez ef&#x00ea;meras,
destinadas a pr&#x00f3;ximo desaparecimento ante as exig&#x00ea;ncias
crescentes da civiliza&#x00e7;&#x00e3;o e a concorr&#x00ea;ncia material
intensiva das correntes migrat&#x00f3;rias que come&#x00e7;am a invadir
profundamente a nossa terra.
       </P>
```

```
      <P>   O jagun&#x00e7;o destemeroso, o tabar&#x00e9;u ing&#x00ea;nuo e o
caipira simpl&#x00f3;rio ser&#x00e3;o em breve tipos relegados &#x00e0;s
tradi&#x00e7;&#x00f5;es evanescentes, ou extintas.
      </P>
</DOCUMENT>
```

FIGURA 2.11 – LIGAÇÃO DA FOLHA DE ESTILO *ESTILO.CSS* COM O DOCUMENTO XML *SERTOES.XML*

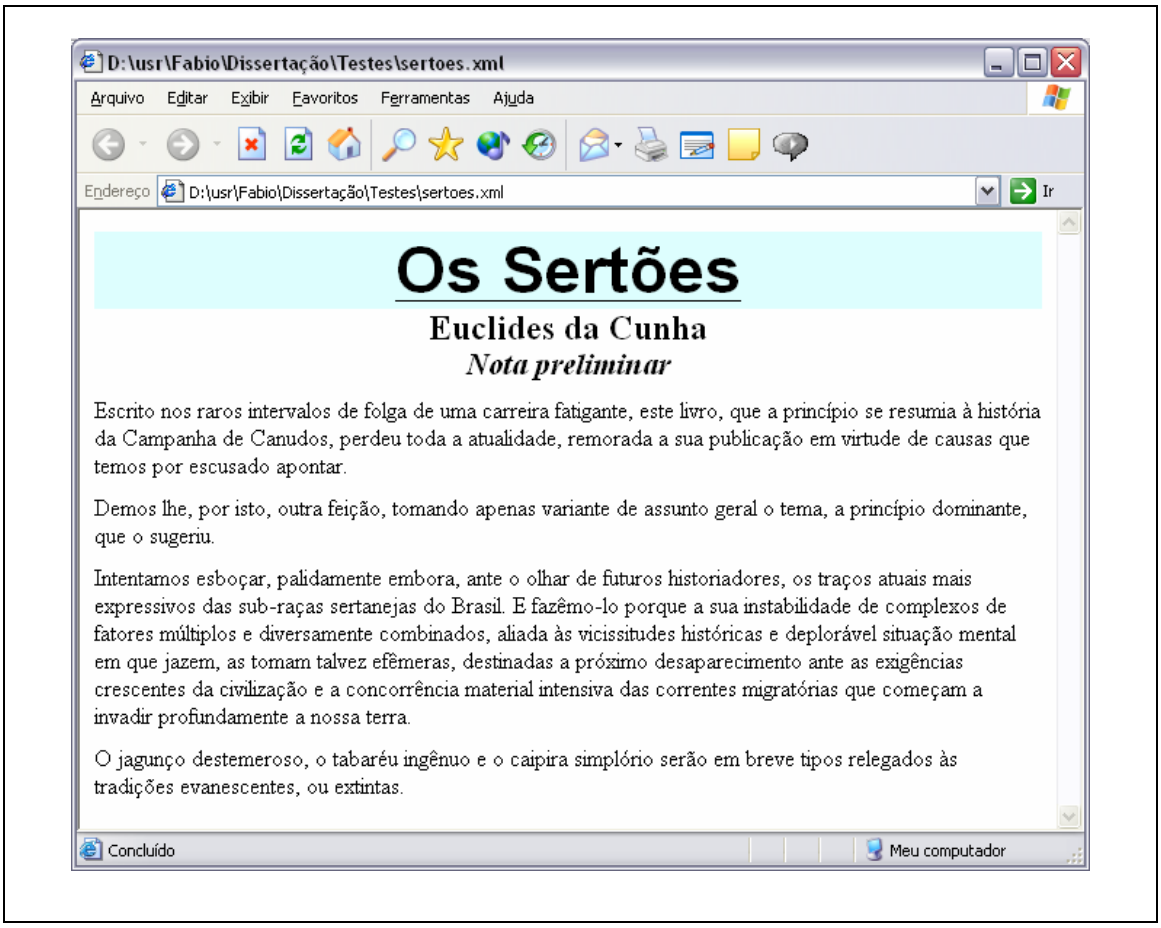

FIGURA 2.12 – *BROWSER* MS-INTERNET EXPLORER APRESENTANDO DOCUMENTO XML FORMATADO POR FOLHA DE ESTILO CSS

2.4.7.2 XSL

A abordagem CSS é ideal para controlar como os diferentes elementos de um documento são apresentados, mas não provê um controle satisfatório sobre o que será exibido e em que ordem. Para tratar essa carência o W3C definiu XSL (*Extensible Style*

Language), uma linguagem que permite transformar documentos XML, estando apta a adicionar, remover ou reordenar elementos desses documentos [NAKHIMOVSKY 00].

De modo geral, XSL foi projetada para controlar a produção de um documento XML para a tela, página impressa ou qualquer outro esquema de exibição bidimensional. A filosofia da XSL baseia-se na "ligação tardia" do conteúdo de um documento e sua "*renderização*"[22], que preferencialmente deve ocorrer no momento da apresentação do documento [MCGRATH 98]. A fig. 2.13 mostra a interação que ocorre entre o documento XML e as folhas de estilo, definindo que tipo de saída será produzida [MCGRATH 98].

A abordagem de "ligação tardia" entre conteúdo/apresentação adotada pela XSL possui benefícios significantes, tais como [MCGRATH 98]:

- Incremento de produtividade e capacidade de manutenção, pois, a aparência de um documento (ou milhares de documentos) pode ser facilmente modificada simplesmente alterando-se uma única folha de estilo;

- Flexibilidade, sendo permitidas múltiplas formas de apresentação baseadas em um único documento fonte. São possíveis, até mesmo, rearranjos do conteúdo para a geração de visões diferenciadas da informação;

- Facilidade ancorada na independência existente entre o conteúdo de um documento e sua forma de apresentação. Operações tais como pesquisa, coleta ou rearranjo do conteúdo podem ser executadas sem preocupação pertinente à como a informação de formatação estaria mesclada ao conteúdo.

[22] Produzir uma imagem gráfica a partir de um arquivo de dados em um dispositivo de saída, como um monitor de vídeo ou impressora

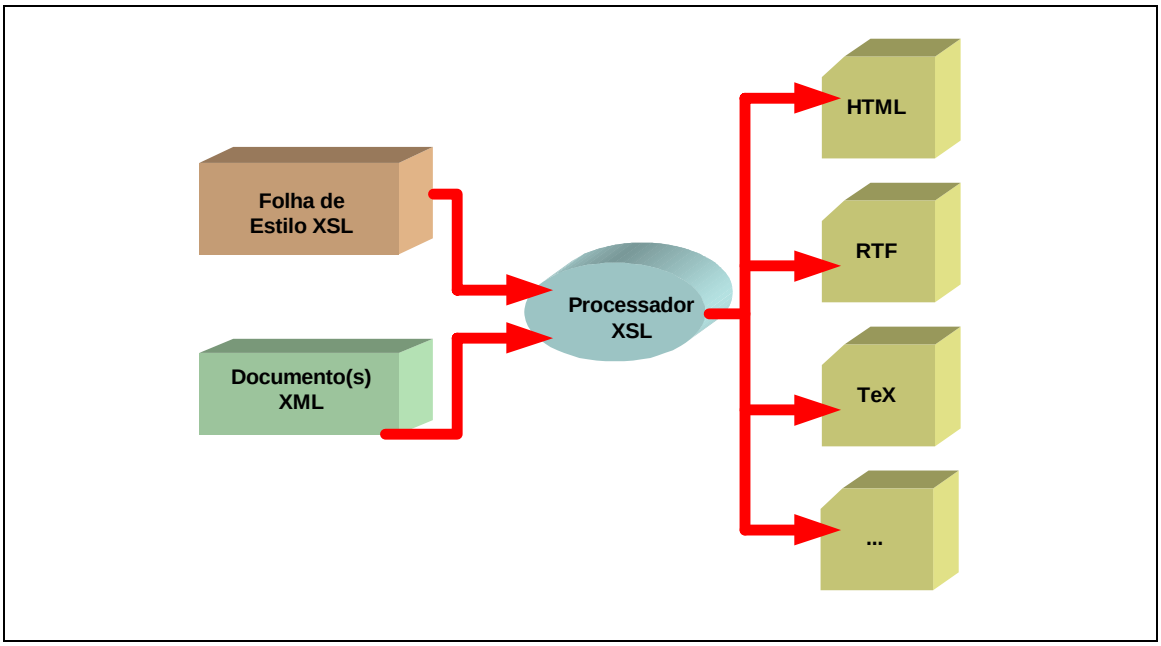

FIGURA 2.13 – INTERAÇÃO DO DOCUMENTO XML COM AS FOLHAS DE ESTILO XSL

Apesar folhas de estilo CSS serem bastante populares, principalmente no meio HTML, a XML é mais flexível e poderosa, sendo as próprias folhas de estilo XSL documentos XML bem formados. Ao contrário da CSS, que permite apenas a definição do formato e posicionamento dos elementos, a XSL pode reordenar completamente os elemento de um documento, exibindo ou ocultando, selecionando estilos com base nos elementos, seus atributos e localização. A *core expression language* da XSL é extremamente flexível, baseada na linguagem *Scheme* (um dialeto da linguagem LISP), possuindo recursos para cálculos, teste de condições e outros.

A semântica da maior parte dos elementos de XSL é baseada em uma combinação da DSSSL (*Document Style Semantics and Specification Language)*, definida pelo padrão ISO/IEC 10179:1996 e das CSS.

Basicamente, a XSL pode ser tratada como a união de duas linguagens, que podem ser empregadas de forma independente, a linguagem de transformação e a linguagem de formatação.

A linguagem de transformação é normalmente referenciada como XSLT (*XSL Transformations*) e tornou-se uma recomendação W3C desde novembro de 1999[23]. Contudo, como é uma especificação relativamente recente e ainda em desenvolvimento, o suporte XSLT oferecido pelos muitos dos *browsers* é limitado.

As marcações de um documento XML podem ser inteiramente manipuladas com a XSLT. O processo de transformação XSLT envolve dois documentos: o documento a ser transformado e a folha de estilo que determina as transformações a serem aplicadas.

A transformação em si pode ocorrer de três modos distintos:

- *Server side*. Uma aplicação executando no servidor, como um *servlet* java, pode aplicar automaticamente uma folha de estilo transformando um documento e fornecendo esse documento transformado aos clientes.

- *Client side*. Uma aplicação cliente, como um *browser*, pode realizar a transformação, lendo a folha de estilo e aplicando-a ao documento que está sendo carregado para exibição.

- *Stand alone*. Aplicações independentes (não embutidas num *browser*), geralmente escritas em java, efetuam a transformação.

XSL é composta por três áreas modulares:

a) padrões XSL: define uma sintaxe de padrões, os quais identificam nodos dentro de um documento XML, fornecendo um recurso semelhante à cláusula *where* da Structured Query Language (SQL) definida no padrão ISO/IEC 9075:1999.

b) *"xsl" namespace* : as transformações de XSL são expressas como um espaço de nomes XML, o qual controla a materialização dos resultados da consulta como um documento XML;

c) *"fo" namespace* : os recursos de formatação de XSL são expressos como uma gramática XML.

[23] Disponível em *www.w3.org/TR/xslt*

A sintaxe dos padrões XSL é simples, embora forneça poderosos recursos de consulta. O seu objetivo é identificar um subconjunto de um documento XML, baseado em cadeias de ascendência, caracteres coringa (*wildcards)* e qualificadores, tais como os valores de testes dos atributos. A sintaxe é concisa e modelada por meio de métodos familiares de navegação em diretórios. Sendo uma sintaxe baseada em *strings*, ela reside dentro dos valores dos atributos, linguagens de script e URLs.

O *namespace xsl* define um conjunto de comandos que permite a materialização dos resultados da consulta como XML – na mesma gramática, em uma gramática diferente ou em uma gramática de apresentação, tal como HTML. Um documento XSL contém elementos do *namespace xsl*, que podem:

a) definir um modelo (*template*) para construção de uma parte do documento de saída;

b) preencher o modelo com dados da fonte XML;

c) associar esse modelo a uma consulta;

d) agrupar um conjunto dessas associações consulta-modelo juntos para aninhar um processamento recursivo

e) filtrar e classificar os resultados das consultas.

Os recursos de formatação de XSL são expressos por uma gramática XML – o *namespace fo* – totalmente independente do processo de transformação.

2.4.8 X*link*s e Xpointers

Um dos fatores de sucesso naturais da *Web* está na sua capacidade natural de ligação de recursos. Porém, a estrutura de ligação simples e unidirecional da *Web* não é suficiente para atender os novos requisitos advindos de XML. Isso motivou o W3C a propor uma solução de ligação em XML denominada *XLink* (*XML Linking Language*).

As especificações XPointer e X*Link*s, atualmente em desenvolvimento, introduzem um modelo de ligação para XML. Em X*Link*, um *link* expressa um relacionamento entre recursos, que podem ser qualquer local (um elemento, o seu conteúdo, ou uma parte do seu conteúdo, por exemplo) que é endereçável em *link*. A natureza exata do relacionamento entre os recursos dependente da aplicação que processa o *link* e da informação semântica fornecida. O objetivo do XPointer é apontar para o elemento correto no documento XML alvo.

Os *link*s XML permitem que o usuário controle a conduta muito mais estritamente e fornecendo suporte para *links* de mão dupla e multidirecionais por meio do conceito *out-of-line*.

A XML-*Link* foi construída a partir de idéias do HTML, por isso não há nem um problema em incluir um *link* a um documento XML dentro de uma página HTML. Ambos os *link*s têm a mesma aparência conforme mostrado abaixo:

```
  <a href=http://www.anothersite.org/pub/doc3.xml>
 <a href=http://www.anothersite.org/pub/doc3.xml#section2>
```

A especificação do atributo HREF da XML é projetada para interoperar com a HTML. Ela consiste em uma URL, opcionalmente seguida de ? ou uma *query*, ou ainda um # ou / e um identificador fragmentado, cuja interpretação depende do tipo de recurso referido pelo URL.

Dentre as características destacáveis do XML-*link* pode-se citar a facilidade de qualquer elemento indicar a existência de um *link*, fornece rótulos inteligíveis para máquinas e seres humanos.

Especificando condutas para quando os *link*s são exibidos e processados ou ainda atravessados.

Aceitando grupos de *link*s estendidos e com canais múltiplos finais.

Uma aplicação importante está em sistemas de hipertexto (*hypertext*).

X*Link*s podem residir dentro ou fora dos documentos em que os recursos residam.

2.4.9 Análise de Dados XML:SAX e DOM

A chave para processar qualquer dado XML é o *parser* XML, o software que lê dados XML e disponibiliza tais dados para outros softwares. O *parser* provê uma API que permite ao programador adicionar código para manipular os dados, geralmente estruturados como uma árvore [GRAHAM 00], [NAKHIMOVSKY 00].

Existem basicamente dois tipos de *parsers* XML: validadores e não-validadores. Os validadores lêem tanto o documento XML quanto o DTD associado, testando a estrutura de marcação do documento com relação às regras gramaticais definidas no DTD. Documentos inconsistentes com o DTD serão recusados e o processamento abortado. Já os parsers não-validadores não utilizam DTD e não validam o documento XML, apenas checam se o documento é bem formado. Tecnicamente, um parser não-validador não ignora completamente o conteúdo de um DTD associado a um documento XML. Na verdade, o *parser* não-validador processará o conteúdo do DTD que ele puder, ignorando apenas as partes que ele não consiga analisar.

Normalmente *browsers Web* que podem processar XML possuem *parsers* não-validadores embutidos ou senão são equipados com *parsers* que podem trabalhar tanto no modo validador quanto não-validador.

Dada a existência do *parser* XML, surge a problemática da integração do mesmo com a aplicação XML, que realmente fará uso dos dados XML disponibilizados pelo *parser*.

Duas interfaces de programação vêm polarizando a escrita de *parsers* XML, sendo cada uma dessas soluções voltada para uma classe específica de problemas. Tais interfaces são SAX (*Simple Application programming interface for Xml*) e DOM (*Document Object Model*).

SAX foi projetado por David Megginson[24], fora do W3C. SAX é simples, constituindo uma ferramenta comum para processamento XML, sendo amplamente suportado por vários softwares XML comerciais e não-comerciais. A filosofia do SAX

[24] www.megginson.com

está baseada em eventos: o *parser* reporta eventos diretamente para a aplicação através de funções de *callback* registradas. As funções de *callback* atuam como manipuladores de evento (*event handlers*) para diferentes eventos, similarmente ao que ocorre com manipuladores de evento em interfaces gráficas que respondem às ações do usuário.Como ilustração, a fig. 2.14, detalhe (a), exibe um documento XML bastante simples e a fig. 2.14, detalhe (b), a seqüência de eventos reportados por uma aplicação baseada em SAX.

```
(a)
<?xml version=”1.0” encoding=”isso-8859-1”?>
<PRINCIPAL>
<PARTE> Uma parte </PARTE>
<ÚNICO/>
</PRINCIPAL>
(b)
Evento: Início do documento
Evento: Elemento inicial: <PRINCIPAL>
Evento: texto string [\0a]
Evento: texto string [ ]
Evento: Elemento inicial: <PARTE>
Evento: texto string [Uma parte]
Evento: Elemento final: <PARTE>
Evento: texto string [\0a]
Evento: texto string [ ]
Evento: Elemento inicial: <UNICO>
Evento: Elemento final: <UNICO>
Evento: texto string [\0a]
Evento: Elemento final: <PRINCIPAL>
Evento: Fim do documento
```

FIGURA 2.14 – UM DOCUMENTO XML E SEQÜÊNCIA DE EVENTOS SAX

A abordagem orientada a eventos do SAX é útil para processamento de uma única passagem no documento XML, mas não é ideal para manipulação repetitiva dos dados do documento em memória. Nesse aspecto, o DOM[25], padronizado pelo W3C, especifica um meio para tratar um documento XML como uma árvore de nós carregada em memória para processamento. No modelo DOM, cada item de dado discreto é considerado um nó e os elementos filho ou o texto incluído tornam-se sub-nós

[25] www.w3.org/DOM

[HOLZNER 01]. DOM permite a reconstrução completa de um documento XML de entrada e provê acesso a qualquer parte do documento, suportando métodos padronizados manipular e modificar o documento original.

Claramente, o DOM é mais poderoso que SAX, contudo aplicações baseadas em DOM tendem a consumir mais recursos de memória do computador e executam mais lentamente quando comparadas a aplicações SAX [GRAHAM 00], principalmente se o documento XML de entrada for grande, visto que DOM mantém o documento inteiro em memória.

É interessante notar que uma abordagem pode ser implementada utilizando os recursos fornecidos por outra: eventos SAX podem ser gerados pelo caminhamento em um objeto DOM assim como um objeto DOM pode ser gerado pela manipulação apropriada de eventos SAX [NAKHIMOVSKY 00].

2.4.10 Consulta a Documentos XML com *XPath*

Os *parsers* discutidos na seção anterior proporcionam um meio básico para acessar informações em documentos XML. Contudo, para muitas aplicações necessita-se de um método mais poderoso para extração dessas informações. Assim, *XPath* provê uma linguagem estendida de consulta para extrair partes de documentos XML. As consultas *XPath* descrevem um “caminho” na estrutura de árvore gerada pelo *parser* DOM a partir do documento XML original.

Para exemplificar, a consulta */descendant::ds::Service[@rdf:about=ID]* seleciona uma certa descrição de elemento em um documento XML. O eixo *descendant* indica que a pesquisa deve ocorrer em qualquer direção abaixo do nó atual (nesse caso o nó raiz). Após *::* segue-se o nome do nó desejado, nessa consulta *Service*. Finalmente, envolto por colchetes tem-se um predicado de filtro, indicando para selecionar apenas nós com o atributo *rdf:about* que tenham um valor *ID*.

2.4.11 Padrões de Mercado Oriundos de XML

XML é uma sintaxe de baixo nível para representação de dados estruturados. Logo, essa simples sintaxe pode ser empregada para suportar uma grande variedade de aplicações específicas. Atualmente, existe uma série de padrões de mercado, desenvolvidos a partir de XML, visando estabelecer um formato único e uma interface padronizada para troca de documentos entre diferentes instituições pertencentes aos mais diversos segmentos de atividade [FURGERI 01]. Como não há um controle efetivo, muitos acrônimos foram adotados por segmentos mercadológicos distintos, gerando conflitos, como por exemplo MML.

Fora do âmbito da W3C, muitos grupos também se dedicam à definição de novos formatos para troca de informações. O número de aplicações XML cresce rapidamente, em diversas áreas como finanças, assuntos governamentais, telecomunicações e medicina, dentre outras. Alguns dos padrões definidos são:

- MathML – *Mathematical Markup Language* utilizada para descrição de fórmulas matemáticas;
- SVG - *Scalable Vector Graphics* define um conjunto de delimitadores XML para gráficos, projetados para inclusão em HTML e outros documentos. De modo similar, outros conjuntos de delimitadores padrão XML para aplicações específicas também são candidatas a inclusão em documentos HTML;
- MML – *Medic Markup Language* utilizada para a troca de informações entre diferentes instiuições;
- MML – *Music Markup Language* para eventos musicais;
- SMIL - Linguagem de Integração de Multimídia Sincronizada (SMIL) é uma aplicação XML consistindo de uma linguagem declarativa para planejar apresentações multimídia pela *Web*;
- CML – *Chimical Markup Language* para descrição textual e gráfica de fórmulas químicas;
- ComicsML – *Comics Markup Language*, padrões de criação de histórias em quadrinhos;

- AML - *Astronomical Markup Language* para padronização de elementos astronômicos;

- RDF - *Resource Description Format* desenvolvido pelo grupo de trabalho de atividades em metadados da W3C. Ele utiliza um modelo de dados simples expresso em sintaxe XML como a base de uma linguagem para representar propriedades de recursos na *Web*, como imagens, documentos e relacionamentos mantidos entre estes. A Plataforma para Seleção de Conteúdo na Internet (PICS) foi redefinida em RDF. A estrutura da PICS fornece um método para inserção de rótulos a diferentes materiais (indicando, por exemplo, se aquele conteúdo é apropriado para crianças).

3. Linguagem Java

Java é uma linguagem concisa e portável, implementando recursos de imagens gráficas, GUI (*Graphical User Interface*), tratamento de exceção, *multithreading*, multimídia, acesso a banco de dados, computação distribuída e rede cliente/servidor baseada na Internet e *World Wide Web*(WWW). Java vem se solidificando no mercado como uma das principais linguagens de programação de propósito geral [DEITEL 01]. Linguagens que fornecem suporte a programação orientada a objetos (OO) estabeleceram uma posição de destaque em meados da década de 90. Java é uma das linguagens recentes projetadas para suportar OO [ECKEL 01], que apesar de não aceitar outros paradigmas, ainda emprega algumas estruturas básicas e possui similaridades com as linguagens imperativas das quais descende [SEBESTA 00].

3.1 Origens

Java, similarmente ao que aconteceu com uma série de outras linguagens de programação, foi projetada para ser aplicada em um domínio de problema que parecia não possuir ainda uma linguagem de programação adequada [SEBESTA 00]. Esse domínio consistia em sistemas embutidos (*embedded systems*) para dispositivos eletrônicos de consumo em massa [GOSLING 96], como fornos microondas e sistemas de TV interativos. A confiabilidade era considerada uma questão importante para o software dos produtos eletrônicos de consumo. Reconhecendo o grande potencial dessa área, em 1991 a *Sun Microsystems* financiou uma pesquisa corporativa interna com o codinome *Green* [DEITEL 01]. Inicialmente, a empresa considerou o uso de linguagens já tradicionais como C e C++, mas nenhuma delas mostrou-se inteiramente satisfatória para a implementação dos softwares de dispositivos eletrônicos: C era pequena, porém sem suporte a orientação a objetos e C++ era orientada a objetos, porém excessivamente grande e complexa [SEBESTA 00].

Norteando-se nos princípios de confiabilidade, simplicidade e suporte a OO [SEBESTA 00], o projeto *Green* propiciou o desenvolvimento de uma linguagem

batizada pelo projetista-chefe James Gosling[26], em homenagem ao carvalho que podia ser visto da janela de seu escritório na *Sun*, como *Oak* [GOSLING 96]. Mais tarde descobriu-se a existência de uma outra linguagem também chamada *Oak*. Então, uma equipe da Sun que visitava um *café* local sugeriu o nome Java[27] referenciando a cidade de origem de um café importado [DEITEL 01].

Após atravessar um período de dificuldades [DEITEL 01], o projeto *Green* encontrou na *World Wide Web* um vasto campo para aplicação da novata linguagem Java : o desenvolvimento de páginas *Web* com conteúdo dinâmico. A *Sun* apresentou formalmente a versão alfa da linguagem em maio de 1995 em uma renomada conferência.

Mais tarde, em janeiro de 1996, a versão 1.0 de Java foi liberada. Quase um ano depois, em dezembro de 1996, uma nova versão, denominada 1.1, foi disponibilizada e englobava vários recursos como acesso a bancos de dados (*JDBC*), chamadas remotas (*RMI*), serialização e uma arquitetura de componentes (*Java Beans*).

3.2 Características

Segundo a Sun Microsystems [FLANAGAN 97], Java pode ser definida como uma “linguagem simples, orientada a objetos, com tipagem forte, interpretada, independente de plataforma, *multithreaded*, com coleta de lixo, robusta, segura, extensível e estruturada”.

3.2.1 Simples

Java apresenta grande similaridade com outras linguagens bastante difundidas. A semelhança sintática com C/C++ é evidente, enquanto o modelo de objetos e de

[26] James Gosling, projetista-chefe da linguagem Java, era então conhecido por ter projetado o editor UNIX *emacs* e a interface de janelas *NeWs*

[27] Java é uma metonímia usada para referenciar o café nos EUA, tendo nascido quando a importação de café para o país provinha principalmente da Indonésia e uma célebre ilha vulcânica da Indonésia chama-se Java.

execução foi baseado em *Smalltalk* e *Simula 67* [SEBESTA 00]. Tais aspectos fazem com que um programador já familiarizado com qualquer uma dessas linguagens tenha maior facilidade e rapidez para aprender Java.

Várias outras linguagens mais poderosas que C++ foram projetadas nos últimos anos, mas não atraíram a mesma atenção. Um grande número de empresas e desenvolvedores utilizou e ainda utiliza C/C++, existindo uma grande comunidade em torno da linguagem. Tal fato levou a Sun a selecionar a linguagem como modelo sintático/semântico para Java, de modo que um programador C/C++ pudesse dominar Java em um espaço de tempo mais reduzido. Esta característica é provavelmente uma das principais razões que ensejaram a popularidade da linguagem.

Apesar das semelhanças, Java não herdou a maioria das complexidades de C++. Todos os recursos de C++ considerados desnecessários pela Sun (como *structs*, *unions* e *typedef*) foram deixados de lado. Assim, Java não possui ponteiros explícitos, arquivos de cabeçalho, pré-processadores, estruturas, uniões, matrizes multidimensionais, gabaritos nem sobrecarga de operadores.

3.2.2 Orientada a Objetos

Java é uma verdadeira linguagem de programação orientada a objetos. Tudo é manipulado como objeto em Java. A única exceção é os tipos simples e primitivos como números e variáveis *booleanas*, que não são objetos por questões de desempenho. Mesmo assim, podem ser encapsulados em objetos quando desejável.

Uma diferença fundamental entre outras linguagens atuais com suporte a orientação a objetos, como Ada 95 e C++, e Java é que esta última não possibilita a implementação de subprogramas independentes [SEBESTA 00]. Todos os subprogramas são métodos e devem necessariamente estar definidos em classes. Toda variável ou método pertence a uma classe ou objeto e só pode ser invocada através dessa classe ou objeto.

Como dito acima, os programas escritos em Java são organizados em classes, que podem ser instanciadas para produzir objetos. As classes também podem herdar características (métodos e variáveis) de outras classes. Contudo, diferentemente de C++ Java não provê suporte a herança de múltipla de classes, mas permite que uma classe implemente mais de uma interface (ou classe abstrata). A interface pode ser grosseiramente conceituada como sendo um tipo especial de classe que não contém detalhes de implementação. Essa abordagem evita o problema de uma classe herdar comportamentos de classes-base que são redundantes, contraditórios ou mutuamente excludentes.

3.2.3 Interpretada e Independente de Plataforma

Um programa escrito em Java precisa ser compilado antes de ser executado. O compilador traduz o código-fonte e gera arquivos objeto chamados arquivos de classe. Cada programa Java consiste da implementação de uma única classe. Depois de compilado, pode ser executado em qualquer plataforma onde exista um sistema de tempo de execução Java (*runtime*).

A compilação de um programa em C++ realiza a tradução do código-fonte da linguagem em instruções que são interpretadas pelo microprocessador da máquina onde foi compilado. O programa então só roda em outra máquina que tenha o mesmo tipo de processador. Já um programa em Java, quando compilado, gera instruções (*byte-codes*) para um microprocessador virtual. Existem implementações desse microprocessador virtual para várias plataformas e o programa então rodará em qualquer uma delas. A fig. 3.1 exibe de maneira esquemática o processo de compilação de um programa Java.

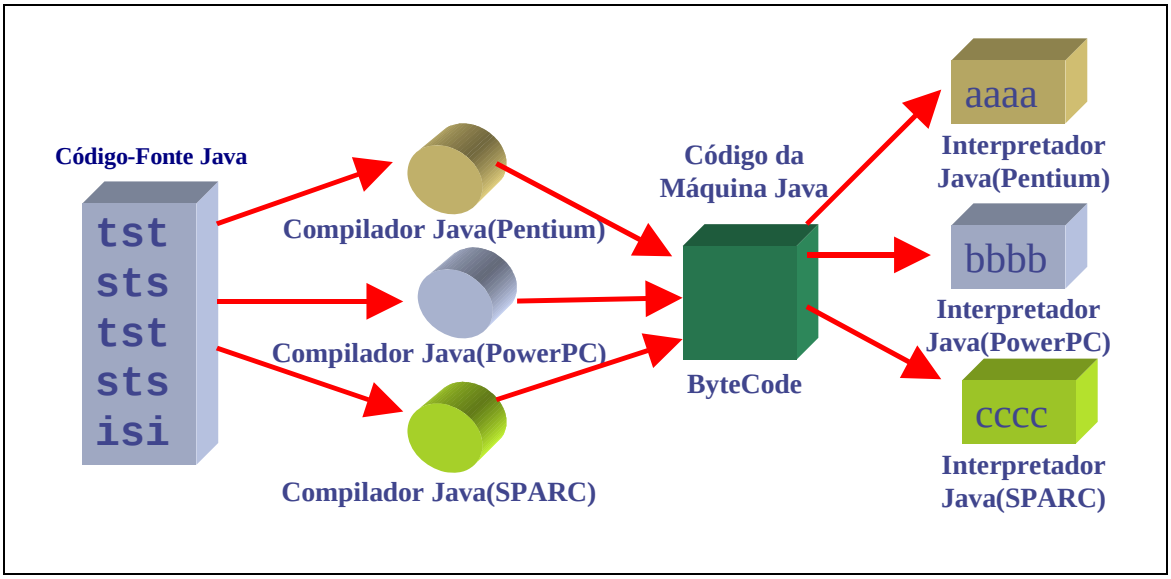

FIGURA 3.1 – ESQUEMA DE COMPILAÇÃO JAVA

Por ter suas instruções interpretadas por um software (processador virtual), os programas em Java são mais lentos que os escritos em C ou C++. A igualdade é atingida usando microprocessadores virtuais com compiladores JIT (*Just-In-Time*), que já são bastante comuns. Esse tipo de sistema, converte as instruções em *byte-codes* para instruções do microprocessador na hora da execução, fazendo com que programas escritos em Java não tenham grandes perdas de desempenho em relação a programas escritos em C ou C++.

3.2.4 Robusta

Java é uma linguagem com tipagem de dados forte, isto é, ela exige que os tipos dos objetos e variáveis sejam explicitamente definidos durante a compilação. O compilador não aceita qualquer indefinição em relação ao tipo de dados. Esta característica garante uma maior segurança do código e o torna menos propenso a erros.

Programas em Java não provocam *core-dumps* ou GPFs (*General Protection Fault*). Enquanto programas escritos em C ou C++ podem alterar qualquer posição da memória do computador, os programas escritos na linguagem Java não têm acesso direto à memória e deixam o controle a cargo do sistema operacional.

O sistema de gerenciamento de memória de Java dispensa os programadores da tarefa de liberação da memória usada. Um processo chamado de coletor de lixo (*garbage collector*) opera sempre em uma *thread* de baixa prioridade em *background*, liberando automaticamente recursos do ambiente que não são mais utilizados.

Caso ocorram erros ou situações inesperadas, Java possui um meio de lidar com elas, recuperando-se do erro se possível. O controle de exceções (condições excepcionais) é uma parte básica da linguagem e, em muitos casos, seu uso é obrigatório.

Essa gama de recursos confere maior robustez aos programas codificados em Java, tornando os erros menos freqüentes e permitindo a detecção deles nos primeiros estágios do desenvolvimento. Isto resulta em um custo menor de desenvolvimento, redução do número de *bugs* e aplicações mais confiáveis.

3.2.5 Dinâmica

A linguagem Java foi projetada para se adaptar a um ambiente dinâmico, em constante evolução. Possui uma representação de tempo de execução que permite ao programa saber a classe a que pertence um objeto, no momento em que o recebe, durante a execução. Isso permite a inclusão dinâmica de classes que podem estar armazenadas em qualquer máquina remota.

Java também suporta a integração com métodos nativos de outras linguagens. Desta forma, podem ser implementados aplicativos híbridos em Java e C++, aproveitando o grande volume de código existente atualmente em C++.

3.2.6 Segura

Por ter uma tipagem de dados forte, somente permitir acesso aos campos membro pelo nome (e não por endereço), não ter aritmética de ponteiros e nem qualquer tipo de acesso direto às posições de memória, um programa compilado em Java pode ser

totalmente verificado antes de ser executado. A verificação dos byte-codes é realizada nos browsers *Web* que suportam Java para garantir que as applets não estejam violando as restrições da linguagem e não possam provocar danos no computador do usuário.

Depois da verificação, as *applets* podem ser otimizadas pelo sistema de execução, para garantir um melhor desempenho. A execução na máquina do cliente também é restrita, não permitindo a escrita no disco nem o acesso a outra localidade na rede, excluindo-se aquela que enviou a *applet*. Como Java não fornece acesso direto à memória, torna-se muito difícil o desenvolvimento de vírus e outros programas maliciosos, da forma como são feitos hoje, usando somente a linguagem Java.

3.2.7 Multi-linha (multithreaded)

Programas em Java podem ter mais de uma linha de execução (*thread*) ocorrendo ao mesmo tempo. Os programadores podem definir quando e com que prioridade certas linhas de execução serão executadas. A maior vantagem dessa característica é a possibilidade das aplicações realizarem processamento em *background* enquanto os usuários interagem a interface das mesmas.

Java ainda fornece meios de sincronizar as linhas de execução. Quando um programa referencia várias delas e há o risco de utilizarem o mesmo conjunto de dados, é necessário sincronizar as ações para que não ocorram conflitos. Em Java, um método pode ser declarado sincronizado para garantir que o objeto no qual atua não possa ser alterado por outros métodos enquanto estiver operando sobre ele.

3.3 Considerações Finais do Capítulo

Nesse capítulo foram abordadas, de maneira não exaustiva, as características gerais da linguagem de programação Java. Visto que o foco da linguagem Java está na computação distribuída [ECKEL 01], redes e *Web* [DEITEL 01], [SEBESTA 00], justifica-se sua escolha para aplicação no presente trabalho.

4. Objetos Distribuídos e a Arquitetura CORBA

4.1 Objetos Distribuídos

Um objeto distribuído é essencialmente um componente, uma peça de software provida de “inteligência auto-contida”, que pode interoperar com outros objetos distribuídos através de sistemas operacionais, redes, linguagens de programação, aplicações, ferramentas e dispositivos diversos. Existe recentemente uma tendência de se construir sistemas computacionais abertos utilizando objetos distribuídos como infraestrutura. Tais sistemas possuem diversas vantagens sobre os sistemas tradicionais sob diversos pontos de vistas, conforme ressaltado em [ORFALI 99]. Os usuários recebem suas aplicações personalizadas, somente com os recursos necessários, evitando, assim, o denominado *fatware*; projetistas podem montar sistemas rapidamente utilizando "objetos de prateleira" existentes; e distribuidores podem vender aplicações com recursos específicos para determinado tipo de mercado, por exemplo, editores de texto para advogados [ORFALI 99].

Além disso, esses objetos distribuídos possuem as mesmas características representativas dos objetos das linguagens de programação: encapsulamento, polimorfismo e herança, tendo, dessa forma, as mesmas vantagens principais: fácil reusabilidade, manutenção e depuração.

Em [HARKEY 98] enfatiza-se os benefícios da utilização de objetos em sistemas distribuídos. Por exemplo, sua utilização ajuda a tratar a heterogeneidade de alguns sistemas, pois os serviços fornecidos são separados de suas implementações. Observa-se também que o encapsulamento é adequado para implementar unidades de distribuição (que migram de forma independente), unidades de falha e unidades de segurança.

Entretanto, no mesmo texto são discutidas algumas dificuldades para implementar herança em sistemas distribuídos. A herança permite diferentes objetos compartilharem código (implementação de outro objeto), capacidade que oferece benefícios óbvios de

redução de duplicidade de código. Porém, em sistemas distribuídos, não é possível o compartilhamento de código de objetos em nós separados.

Um outro benefício da utilização de objetos distribuídos que se aplica especificamente em sistemas de tempo real é o polimorfismo de performance (ou polimorfismo temporal). Esse mecanismo permite que uma interface possua várias implementações diferentes do mesmo método, cada uma com um tempo de execução diferente. Dependendo de determinados aspectos temporais, o método que puder atender aos requisitos temporais especificados pelo cliente naquele momento, será executado.

Cada objeto distribuído não opera sozinho. A princípio ele é construído para trabalhar com outros objetos e, para isso, precisa de uma espécie de "barramento". Tais "barramentos" fornecem infra-estrutura para os objetos, adicionando novos serviços que podem ser herdados durante a construção do objeto, ou mesmo em tempo de execução para alcançar altos níveis de colaboração com outros objetos independentes. CORBA (*Commom Object Request Broker Architecture*) pode ser caracterizada como um exemplo de "barramento" que será visto a seguir.

4.2 A Arquitetura CORBA

A arquitetura CORBA começou a ser definida pela OMG (*Object Management Group*) em 1989 [OMG 01].

A OMG é uma organização internacional suportada por centenas de membros, abrangendo um grande espectro de interesses, desde usuários até projetistas de sistemas. Tal organização promove estudos sobre a tecnologia orientada a objetos no processo de desenvolvimento de software e sua carta de princípios inclui o estabelecimento de diretrizes na indústria e especificações de gerenciamento de objetos para fornecer uma estrutura comum para desenvolvimento de aplicações. O objetivo primário é se alcançar sistemas baseados em objetos em ambientes distribuídos heterogêneos com características de reusabilidade, portabilidade e interoperabilidade.

Em 1990, a OMG criou o OMA (*Object Management Architecture*), objetivando fomentar o crescimento de tecnologias baseadas em objetos e fornecer uma infra-estrutura conceitual para todas especificações OMG. O OMA é composto por quatro elementos principais:

1. *ORB (Object Request Broker)*, habilita os objetos enviarem e receberem requisições e, da mesma maneira, receberem respostas a suas requisições, de forma transparente em um sistema distribuído. O ORB é a fundação para se construir aplicações, utilizando objetos distribuídos, com características de interoperabilidade entre aplicações em ambientes heterogêneos ou homogêneos.

2. Serviços de Objetos, é uma coleção de serviços (interfaces e objetos) que suportam funções básicas para usar e implementar objetos.

3. Facilidades Comuns, é uma coleção de serviços que muitas aplicações podem compartilhar, mas que não são tão fundamentais como os serviços de objetos.

4. Objetos de Aplicação, correspondem à noção tradicional de aplicações de usuários que, por esse motivo, não são padronizados pelo OMG.

A fig. 4.1 apresenta uma idéia geral da estrutura e dos elementos que compõe o OMA.

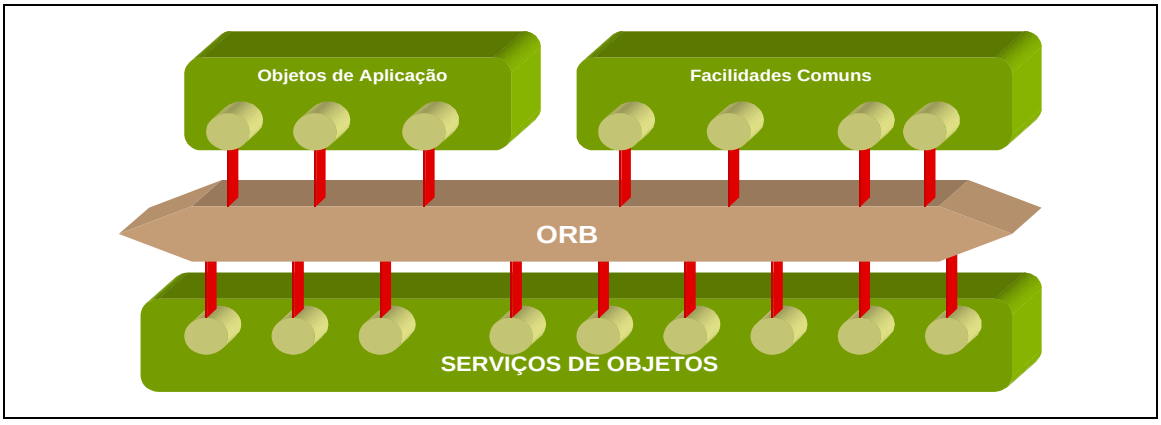

FIGURA 4.1 – ARQUITETURA DE GERENCIAMENTO DE OBJETOS (OMA).

O ORB é o elemento principal desse modelo de referência, pois fornece os mecanismos básicos de envio e de recebimento de chamadas entre objetos e, dessa forma, será descrito com mais detalhes adiante.

4.3 Modelo de objetos CORBA

Para compreender melhor a arquitetura CORBA é necessário conhecer a descrição do modelo de objetos [OMG 01] que fornece conceitos e terminologias de objetos usados pela arquitetura CORBA.

Um conceito básico é o de *sistema de objetos* que é composto por entidades denominadas objetos. Um objeto é uma entidade que fornece serviços aos clientes. Um cliente de um serviço é qualquer entidade capaz de requisitar serviços através de eventos denominados requisições (*requests*). Uma requisição possui informação associada que consiste basicamente da operação, do objeto destino, dos parâmetros e do contexto da requisição.

Um valor é uma instância de um tipo de dados definido no OMG IDL (abordado mais adiante) que pode ser usado como parâmetro em uma requisição. Um valor que identifica um objeto é denominado nome de objeto, e uma referência de objeto é um nome de objeto que denota um objeto em particular.

Uma requisição pode ter parâmetros que podem ser de entrada, saída, ou de entrada e saída, e que são identificados pelas suas posições na requisição. Ela pode ter também um contexto (*context*) que fornece informação adicional sobre a própria requisição. Uma requisição pode retornar um valor de resultado para os clientes, além de retornar os parâmetros de saída. Entretanto, se uma condição anormal ocorrer, uma exceção é gerada. Objetos CORBA podem ser criados e destruídos através de determinadas requisições. O resultado da criação do objeto é disponibilizado ao cliente sob a forma de uma referência de objeto, denotando o novo objeto.

Uma interface é uma descrição de um possível conjunto de operações que um cliente pode requisitar de um objeto. Diz-se que um objeto satisfaz uma interface se ele pode ser especificado como objeto destino em cada operação descrita pela interface. Uma herança de interface fornece o mecanismo de composição para permitir um objeto suportar interfaces múltiplas.

Uma operação é uma entidade que denota um serviço que pode ser requisitado. Uma operação possui uma assinatura que, de um modo geral, descreve os valores válidos dos parâmetros, dos resultados retornados da requisição, a exceção definida pelo usuário que pode ser sinalizada para terminar uma requisição de operação, e a informação de contexto que será fornecida à implementação do objeto. A assinatura descreve também, a semântica de execução da operação no caso de falhas, que pode ser "melhor esforço" ou "no máximo uma vez".

Na implementação de objeto, um método é o código que é executado para fornecer o serviço e a execução de um método é denominada ativação do método.

O modelo de objetos do CORBA é um modelo de objetos clássico, onde um cliente envia uma mensagem a um objeto, o objeto interpreta a mensagem e decide que serviço deve realizar. Como em um modelo clássico, uma mensagem identifica um objeto e zero ou mais parâmetros reais. O primeiro parâmetro é requerido e identifica a operação que deve ser realizada, e serve de base para que, durante a interpretação da mensagem, o objeto receptor (ou o ORB) possa selecionar o método adequado.

4.1.1 Semântica dos Objetos do CORBA

4.1.1.1 Objeto

Entidade identificável e encapsulada que provê serviços que podem ser requisitados por clientes.

4.1.1.2 Requisição

Requisição pode ser conceituada como algo que acontece em um tempo definido (evento). As informações associadas a uma requisição são operação, objeto alvo, parâmetros reais (que podem ser de entrada, de saída ou de entrada e saída) e um contexto de requisição opcional. Uma requisição causa a execução de um determinado serviço, e retorna uma exceção se acontecer uma condição anormal durante a sua execução.

4.1.1.3 Referência a objeto

Valor que identifica um objeto e que seguramente denota um (e somente um) objeto particular. Em contrapartida, um objeto pode ser denotado por diversas referências.

4.1.1.4 Criação e destruição de objetos

Acontecem como resultados de requisições. Do ponto de vista do cliente, não existe nenhum mecanismo especial para criação e destruição de objetos.

4.1.1.5 Tipos

Conceito tradicional de tipos. Formalmente, tipo é entidade identificável associada a um predicado definido sobre os valores membros deste tipo. Em particular, um tipo objeto é um tipo cujos membros são objetos.

Os tipos existentes no modelo de objetos do CORBA são apresentados na fig. 4.2.

Tipos Básicos :

- inteiros com e sem sinal, 16 e 32 bits;
- números em ponto flutuante de 32 e 64 bits;

- caracteres (seguindo definição ISO Latin-I);
- booleano, assumindo os valores TRUE e FALSE;
- tipo opaco de 8 bits que seguramente não sofre nenhuma conversão durante transferência entre sistemas;
- tipos enumerados, consistindo de seqüência enumerada de identificadores;
- tipo string (array de caracteres de tamanho variável). O tamanho da string está disponível em tempo de execução um tipo "*any*" que pode representar qualquer tipo básico ou construído.

Tipos Construídos:

- registro (*struct*);
- união, consistindo de um discriminador seguido por uma instância cujo tipo é apropriado ao valor do discriminador;
- seqüência (*array* de tipo simples de tamanho variável). O tamanho do *array* está disponível em tempo de execução;
- *array* de tipos simples com tamanho fixo ;
- tipo interface, que especifica o conjunto de operações que uma instância deste tipo deve suportar.

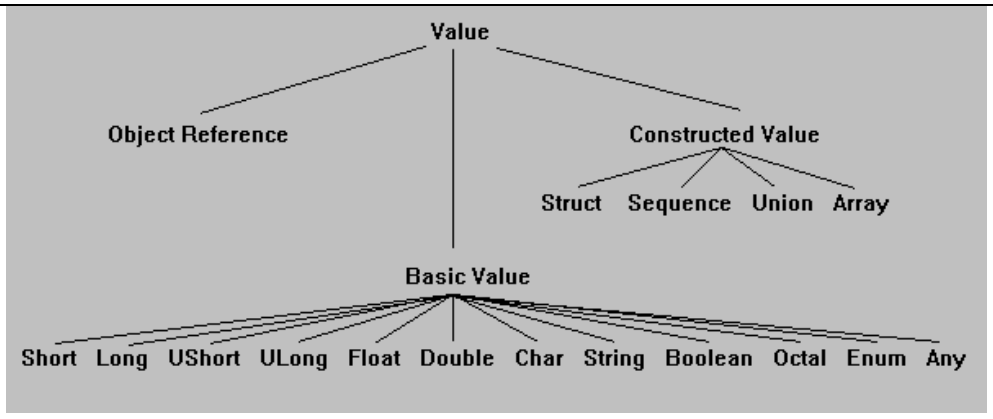

FIGURA 4.2 – TIPOS DO MODELO DE OBJETOS CORBA

4.1.1.6 Interface

Descrição do conjunto de operações possíveis de um objeto que podem ser requisitadas por um cliente. Um objeto pode suportar múltiplas interfaces através de mecanismos como herança. As interfaces são especificadas em uma linguagem denominada IDL.

4.1.1.7 Operação

Entidade identificável que denota um serviço que pode ser requisitado. É extremamente genérica, na medida em que é independente da implementação dos objetos e utiliza-se de mecanismos de herança de interfaces da IDL. Uma operação possui uma assinatura, a qual descreve todos os valores de parâmetros e resultados possíveis, através de:

- especificação de parâmetros necessários em chamadas a esta operação;
- especificação de resultados da operação;
- especificação das exceções que podem ocorrer durante a execução do serviço;
- especificação de informação adicional de contexto;
- indicação da semântica de execução que o cliente espera encontrar na ocasião de uma requisição a esta operação.

A forma geral da assinatura de uma operação é:

```
[oneway] <op_type_spec> <identifier> (param1,...,paramL) [raises
(except1,...,exceptN)]
[context(name1,...,nameM)]
```

onde:

- *oneway* indica a semântica "*best-effort*" de execução, na qual a operação não retorna nenhum resultado, e não há sincronização entre o requisitante e o

término da operação. O *default* é a semântica "*at-most-once*", na qual garante-se que se uma operação retorna com sucesso, esta foi executada exatamente uma vez. No caso de exceção, foi executada uma vez no máximo. A semântica de execução é associada a cada operação, o que garante que tanto o cliente quanto a implementação do objeto sempre assumirão a mesma semântica;

- *op_type_spec* é o tipo do retorno da operação, que é um parâmetro de saída distinto;
- *identifier* é o nome da operação ;
- *parâmetros* são modificados para indicar a direção da informação: *in* indica que a informação passa do cliente para o servidor, *out* do servidor para o cliente e *inout* em ambas as direções ;
- a expressão *raises* é seguida pela lista de exceções definidas pelo usuário (as exceções padrão são incluídas implicitamente) que podem ser sinalizadas para terminar a requisição, indicando que a operação não foi executada com sucesso. Tais exceções podem ser descritas por um registro, no qual são acompanhadas por informação adicional específica da exceção;
- *context* é uma expressão opcional que indica o contexto de requisição disponível na implementação do objeto, e pode afetar a performance da requisição .

4.1.1.8 Atributos

Analogamente aos conceitos de orientação a objetos pura, os atributos de uma interface são logicamente equivalentes a declaração de um par de funções de acesso: uma para recuperação do valor do atributo e outra para alteração do valor (caso o atributo não seja apenas para leitura).

4.1.2 Implementação de Objetos no CORBA

A implementação de um sistema de objetos inclui atividades que podem requisitar serviços adicionais, como por exemplo, cálculo de resultados das requisições e atualização do estado do sistema. O modelo de implementação divide-se em 2 partes: o Modelo de Execução que descreve como os serviços são executados, e o Modelo de Construção que descreve como os serviços são definidos.

4.1.2.1 Modelo de execução

Um serviço requisitado é realizado pela execução de um código que manipula determinados dados e pode alterar o estado do sistema. Tal código é denominado um método. A execução de um método por uma máquina abstrata é chamada de ativação do método.

4.1.2.2 Modelo de Construção

Definem-se os mecanismos para obtenção do comportamento dos pedidos, como definições do estado do objeto, definição dos métodos, definição de como é a infraestrutura do objeto para que ele selecione apropriadamente os métodos a serem executados e definição das ações concretas que devem ser tomadas quando da criação de um objeto, como por exemplo associação do novo objeto aos seus métodos.

4.2 ORB

Objetos clientes requisitam serviços às implementações de objetos através de um ORB (fig. 4.3). O ORB é a unidade que gerencia a transferência de controle e de dados entre os clientes e as implementações, sendo responsável por todos os mecanismos requeridos para encontrar o objeto, preparar a implementação de objeto para receber a requisição, e executar a requisição. O cliente vê a requisição de forma independente de onde o objeto está localizado, qual linguagem de programação ele foi implementado, ou qualquer outro aspecto que não está refletido na interface do objeto. No caso de uma

operação não ser executada com sucesso, gera-se uma exceção que deve ser tratada pelo cliente.

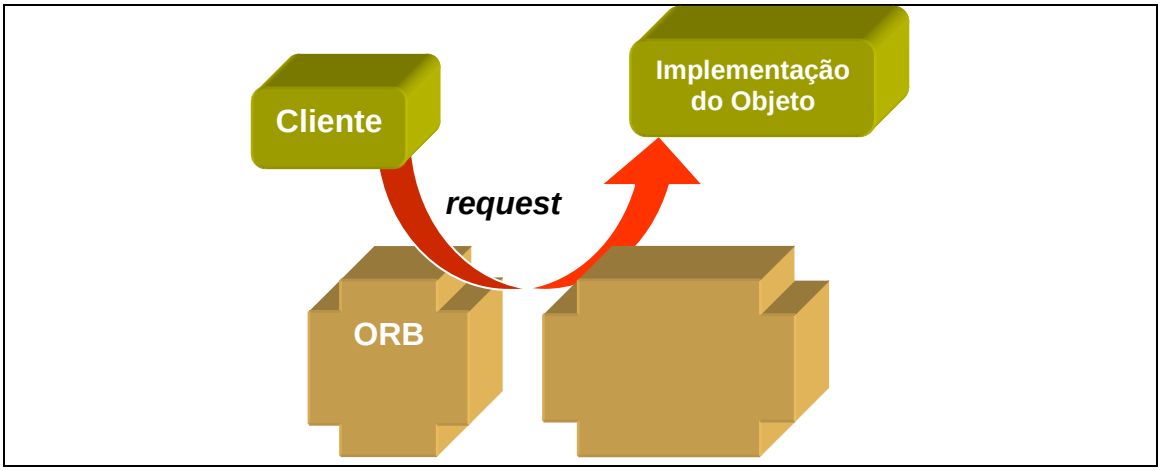

FIGURA 4.3 – CLIENTE ENVIANDO REQUISIÇÃO ATRAVÉS DO ORB.

4.2.1 IDL (Interface Definition Language)

CORBA utiliza a OMG IDL (*Interface Definition Language*) como uma forma de descrever interfaces, isto é, de especificar um contrato entre os objetos. OMG IDL é uma linguagem puramente declarativa[28] baseada em C++. Isso garante que os componentes em CORBA sejam auto-documentáveis, permitindo que diferentes objetos, escritos em diferentes linguagens, possam interagir através das redes e de sistemas operacionais (fig. 4.4) [HARKEY 98].

[28] Uma linguagem declarativa especifica apenas os resultados desejados em vez dos procedimentos detalhados para produzi-los [SEBESTA 00]

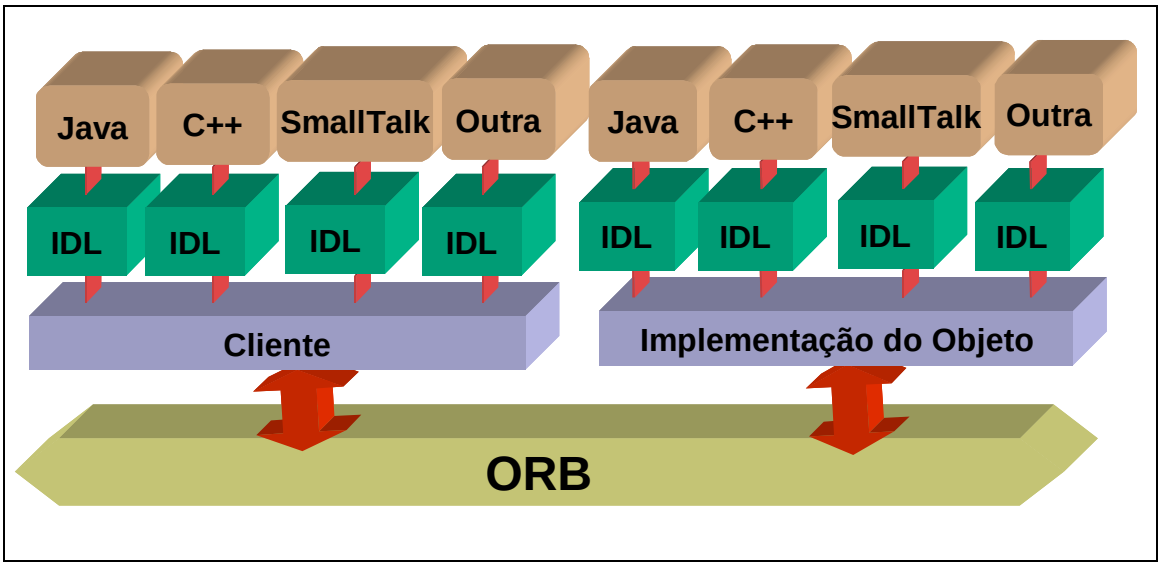

FIGURA 4.4 – IDL PROVÊ INDEPENDÊNCIA DE LINGUAGEM DE PROGRAMAÇÃO ENTRE OS OBJETOS.

Uma definição de interface escrita em OMG IDL define completamente a interface e especifica cada parâmetro da operação. É importante ressaltar que os objetos não são escritos em OMG IDL, que é uma linguagem puramente descritiva. Eles são escritos em linguagens que possuem mapeamentos definidos dos conceitos existentes em OMG IDL. Na especificação CORBA são descritos mapeamentos para C, C++ e Smalltalk, e existem estudos para se adicionar outras linguagens nas futuras versões de CORBA.

De forma geral a IDL obedece as mesmas regras léxicas que a linguagem C++, com apenas algumas palavras reservadas a mais. A gramática IDL é um subconjunto da ANSI C++, com construções adicionais para suportar mecanismos de chamadas a operações. Sendo uma linguagem declarativa, a IDL não possui nenhuma estrutura algorítmica ou variáveis. Para que se evitasse conflito de nomes da especificação CORBA com os da linguagem de programação, convencionou-se que os primeiros devem ser tratados como se tivessem definidos em um módulo denominado CORBA. Os nomes usados na interface devem ser referenciados, portanto, por seu nome completo (*CORBA::<nome>*) .

As diferenças básicas para a sintaxe C++ são restrições como:

- retorno de uma função é obrigatório;
- devem ser fornecidos nomes para cada um dos parâmetros formais na declaração de uma operação;
- uma lista de parâmetros vazia não pode ser substituída pela palavra *void*;
- a declaração dos tipos inteiros quanto a seu tamanho deve ser explícita (*short* ou *long*) ;
- os modificadores *signed* e *unsigned* não são aplicáveis ao tipo *char* .

Existem exceções padrão que são definidas pelo ORB, as quais podem acontecer em qualquer operação e não precisam ser listadas na expressão *raises*. Alguns exemplos de exceções padrão são `UNKNOWN, NO_MEMORY, INITIALIZE, DATA_CONVERSION`,...

Um pequeno exemplo de uma definição IDL:

```
module TProduto {
      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);
      };
```

4.2.2 Serviços

Podemos fazer uma analogia do ORB com uma linha telefônica, que permite enviar e receber mensagens. Entretanto não provê outros serviços que tem que ser definidos a parte e funciona como uma camada externa que utiliza o ORB como "meio". Esses serviços são conhecidos com *Object Services*.

Além desses podemos destacar também *Common Facilities* que também constituí um grupo de serviços de propósito mais geral, enquanto o primeiro é um conjunto de funções básicas.

Esses serviços são definidos em IDL e podem ou não ser implementados em uma linguagem orientada à objetos.

Um serviço é caracterizado pela interface que ele provê, pelos objetos que provêem essas interfaces e por aspectos de qualidade na implementação do serviço que irão influenciar na sua utilização. As interfaces de um serviço são classificadas de acordo com :

1. o tipo de consumidor da interface. Uma interface pode ser utilizada por um usuário final ou por sistema de gerenciamento de funções.

2. o mensageiro da mensagem. O tipo de objeto que apresenta a mensagem. Podem ser objetos específicos, cujo propósito de existência é prover parte do serviço de um *Object Service*, ou objetos genéricos cujo propósito de existência não é prover o serviço correspondente a interface que "carrega", provavelmente herdou esta interface.

Dentre os serviços já identificados e revisados pelo Grupo responsável da OMG, (*Object Service Task Force*) podemos destacar os apresentados na Tabela 4.1.

TABELA 4.1 – SERVIÇOS CORBA

Serviço	**Função**
Archive	Suportar o mapeamento entre objetos ativos e backup.
Backup/Restore	Suporta o backup e recuperação de objetos.
Concurrency Control	Suporta o acesso concorrente a objetos.
Event Notification	Suporta a notificação de eventos a objetos interessados.
Naming	Suporta o mapeamento entre nome e objetos.

Persistence	Suporta a persistência de objetos independente do tempo de vida da aplicação cliente e da implementação que executa os métodos.
Query	Suporta operações sobre conjuntos e coleções retornando conjuntos e coleções.
Relationships	Suporta associação entre dois ou mais objetos.
Replication	Suporta replicação explícita de objetos em um ambiente distribuído e gerencia a consistência das cópias.
Security	Suporta controle de acesso aos objetos.
Time	Suporta a sincronização de relógios em sistemas distribuídos.
Transactions	Suporta transações na execução de operações.

4.2.2 Estrutura do ORB

A fig. 4.5 mostra a estrutura de um ORB, com as setas indicando o fluxo de chamadas que podem ocorrer de clientes para o ORB, e do ORB para as implementações de objetos.

Para fazer uma requisição, um cliente pode usar a Interface de Invocação Dinâmica (DII - *Dynamic Invocation Interface*) ou um *Stub* de IDL. Para algumas poucas e determinadas funções, um cliente pode interagir diretamente com a interface do ORB. A interface do ORB oferece funções que são independentes do adaptador de objetos utilizado, como, por exemplo, funções que executam algumas operações sobre referências de objetos.

O fato de permitir tanto a invocação dinâmica quanto a estática, torna CORBA bastante flexível. A invocação estática, que utiliza *Stubs* de IDL que podem ser gerados

automaticamente através de pré-compiladores, possui uma série de vantagens sobre a invocação dinâmica [HARKEY 98]: é mais fácil de programar, faz verificação de tipos mais robusta, executa mais rápido e é auto-documentável. Já a invocação dinâmica permite a adição de novos serviços às implementações de objetos sem alterações nos clientes. Dessa forma, os clientes podem descobrir em tempo de execução quais são os serviços oferecidos.

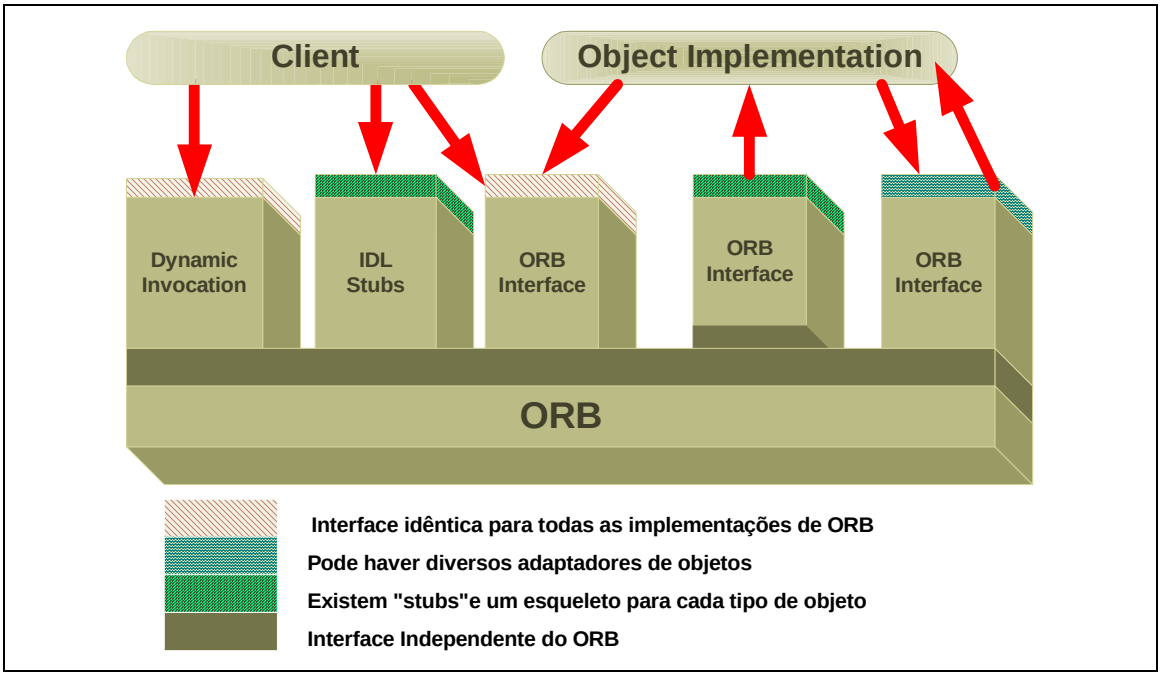

FIGURA 4.5 – ESTRUTURA BÁSICA DE UM ORB.

A invocação dinâmica é possível através dos serviços do Repositório de Interfaces. Um Repositório de Interfaces é uma base de dados que contém interfaces OMG IDL, e seus serviços basicamente permitem o acesso, armazenagem e atualização dessas interfaces.

É importante ressaltar que do ponto de vista da implementação de objetos (ou um *servant*), uma requisição feita estaticamente é indistinguível da feita de forma dinâmica, já que ambas respeitam a mesma semântica da requisição. Uma requisição consiste de uma referência de objeto, uma operação e uma lista de parâmetros.

Ao receber uma requisição, o ORB localiza o código da implementação transmite parâmetros e transfere o controle para a implementação do objeto (*servant*). Quando a requisição é completada, o controle e os valores de saída são retornados ao cliente.

De forma parecida ao que ocorre na requisição de um cliente, o ORB pode invocar a implementação de objeto através de um Esqueleto de IDL Estático ou Dinâmico. A Interface de Esqueleto de IDL Dinâmica (DSI - *Dynamic Skeleton Interface*) é uma forma de se enviar requisições de um ORB para uma implementação de objetos que não possui informações sobre a implementação do objeto em tempo de compilação. O cliente que invoca um objeto, não pode determinar se a implementação está usando um esqueleto de um tipo específico ou DSI para conectar a implementação ao ORB. DSI é útil em soluções de interoperabilidade implementando pontes entre diferentes ORBs, e para suportar linguagens de tipos dinâmicos, como LISP[29] e PROLOG[30].

Para executar a requisição, a implementação de objeto pode obter alguns serviços do ORB através do Adaptador de Objetos. O Adaptador de Objetos se situa no topo dos serviços de comunicação do Núcleo de ORB. Ele fornece o ambiente para instanciar objetos, atribui-los referências de objetos, e passar requisições a eles. Com um adaptador de objetos, é possível a uma implementação de objeto ter acesso a um serviço independentemente se ele está implementado no Núcleo de ORB — se o Núcleo de ORB oferecer o serviço, o adaptador simplesmente fornece uma interface para ele; caso contrário, o adaptador deve implementá-lo no topo do Núcleo de ORB.

O conceito de adaptador de objetos é necessário para permitir que CORBA suporte uma grande diversidade de aplicações, com variados tipos e estilos de implementações de objeto. Se os serviços do adaptador de objetos fossem oferecidos diretamente pelo núcleo do ORB, seria necessária a existência de uma gama muito grande de métodos na interface do ORB para atender todas as demandas dos variados tipos de *servants* existentes.

---

[29] *List Processor*. Primeira linguagem de programação funcional de uso generalizado projetada por John McCarthy, no MIT, entre 1958 e 1959 [SEBESTA 00]

[30] *Programming Logic*. Uma linguagem que provê um método para especificar proposições de cálculo de predicado e uma implementação de uma forma restrita de resolução, sendo o primeiro interpretador implementado em Marseille, França, em 1972 [SEBESTA 00]

Não é necessário que todos adaptadores de objetos ofereçam a mesma interface ou funcionalidade. Algumas implementações de objeto podem ter requisitos especiais, exigindo adaptadores de objetos específicos. Dessa forma, a implementação do objeto pode escolher qual adaptador de objetos utilizar dependendo de quais tipos de serviços ele requer. Entretanto, com o objetivo de tentar conter uma proliferação interminável de tipos de adaptadores de objetos, [OMG 01] recentemente definiu o Adaptador de Objetos Portável (POA - *Portable Object Adapter*). A especificação do POA define um adaptador de objetos que pode ser usado pela maioria dos objetos que possuem implementações convencionais.

Para localizar e ativar implementações de objetos, um ORB utiliza-se de Repositórios de Implementações. Esses repositórios servem adicionalmente para armazenar informações adicionais associadas com implementações de ORBs, tais como, informações de depuração, controles administrativos, segurança, dentre outras.

4.2.3 Interoperabilidade entre ORBs

A especificação de interfaces de objetos obrigatoriamente em OMG IDL, garante a portabilidade dos objetos através de diferentes linguagens, ferramentas, sistemas operacionais e redes. Entretanto, a característica de interoperabilidade entre objetos só foi coberta na versão 2.0 do CORBA, introduzida em dezembro de 1994. Isso se deu através da especificação de uma arquitetura de interoperabilidade, um suporte para pontes entre ORBs, um protocolo para comunicação entre ORBs genérico e um protocolo para comunicação entre ORBs para Internet.

A arquitetura de interoperabilidade do CORBA identifica claramente a regra de diferentes tipos de domínios para informação específica de ORBs. Tais domínios podem incluir domínios de referências de objeto, domínios de tipos, domínios de segurança, dentre outros. Quando dois ORBs estão no mesmo domínio, eles podem se comunicar diretamente. Entretanto, quando a informação em uma invocação deve deixar seu domínio, a invocação deve atravessar uma ponte.

A regra de uma ponte deve assegurar que o conteúdo e a semântica são mapeados adequadamente de um ORB para outro. O suporte para ponte entre ORBs pode também ser usado para prover interoperabilidade com outros sistemas não CORBA.

O protocolo entre ORBs genérico (GIOP - *Generic Inter-ORB Protocol*) especifica uma sintaxe de transferência padrão e um conjunto de formatos de mensagens para comunicação entre ORBs. O protocolo entre ORBs para Internet (*IIOP - Internet Inter-ORB Protocol*) especifica como mensagens GIOP são trocadas usando conexões TCP/IP. Ele especifica um protocolo de interoperabilidade padrão para Internet.

O relacionamento entre GIOP e IIOP é similar ao mapeamento existente entre o OMG IDL e uma linguagem específica: o GIOP pode ser mapeado em diferentes protocolos de transportes, e especifica os elementos de protocolo que são comuns a todos os mapeamentos. Entretanto, o GIOP não fornece uma completa interoperabilidade, da mesma forma que IDL não pode ser usado para se construir programas completos. O IIOP, e outros mapeamentos similares para diferentes protocolos de transportes, são realizações concretas das definições abstratas de GIOP (fig. 4.6).

Os protocolos entre ORBs para ambientes específicos (ESIOP - *Environment-Specific Inter-ORB Protocol*) devem ser usados para interoperar em locais onde uma rede ou infra-estrutura de computação distribuída já esteja em uso. Apesar de cada ESIOP poder ser otimizado para um ambiente em particular, toda especificação ESIOP deve estar conforme as convenções da arquitetura de interoperabilidade para facilitar o uso de pontes, se necessário. O suporte de pontes entre ORBs habilita às pontes serem construídas entre domínios de ORBs que usam IIOP e domínios que usam um ESIOP particular.

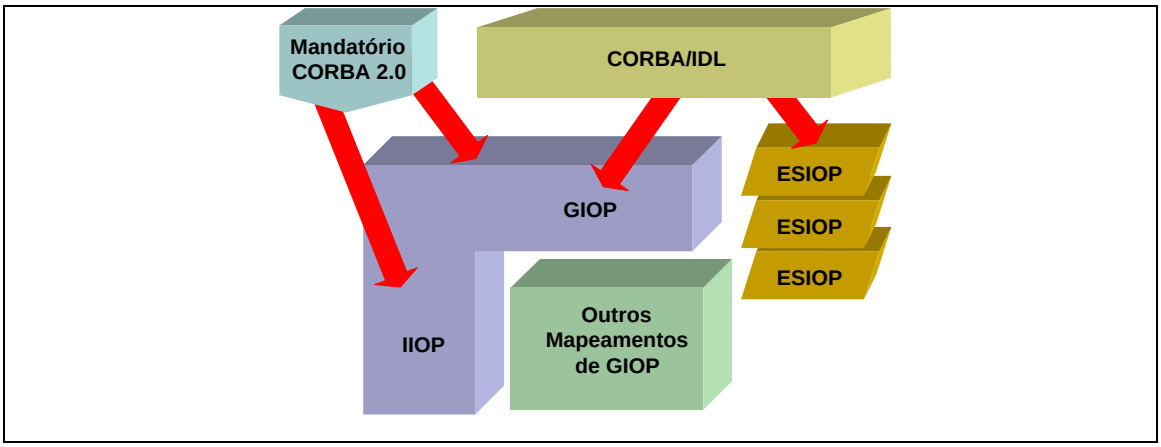

FIGURA 4.6 – RELACIONAMENTOS DE PROTOCOLOS ENTRE ORBS.

4.3 Serviços CORBA

O ORB, por si só, não executa todas as tarefas necessárias para os objetos interoperarem. Ele só fornece os mecanismos básicos. Outros serviços necessários são oferecidos por objetos com interface IDL, que a OMG vem padronizando para os objetos de aplicação poderem utilizar.

- *serviço de nomes*, por exemplo, é basicamente um serviço de localização de objetos, que permite um objeto descobrir outros objetos através de identificadores, ou nomes.

- serviço de controle de concorrência fornece um gerente de "*locks*" que coordena múltiplos acessos a recursos compartilhados, permitindo resolução de conflitos de acesso.

- *serviço de eventos* permite que objetos registrem dinamicamente seus interesses em determinados eventos, possibilitando uma forma de comunicação pouco acoplada e assíncrona entre objetos. Eventos são enviados e recebidos através de um canal de eventos.

- *serviço de tempo* especifica uma interface de serviço de tempo que permite, basicamente, os objetos das aplicações obterem o tempo atual e executarem comparações com o tempo.

- *serviço de tempo* também estende o serviço de eventos, especificando uma interface de serviços de eventos temporizados. Esse novo serviço gerencia objetos manipuladores de eventos temporizados. Um cenário típico de utilização desses serviços é um usuário criar um canal de eventos e registrá-lo como destino dos eventos gerados por um manipulador de eventos temporizados. Dessa forma, o usuário pode usar o objeto manipulador de eventos temporizados para criar eventos temporizados quando desejar. Basicamente, o serviço de eventos temporizados permite que um objeto possa "ligar" o tempo para disparo do evento (indicando inclusive que esse disparo pode ser periódico). Permite também que um objeto que recebeu a notificação de um evento, possa determinar o tempo em que o evento ocorreu.

5 A API AdventSNMP e o Protocolo SNMP

A API AdventNet SNMP é um ambiente que provê acesso multiplataforma . É empregada na implementação de aplicações (ou *applets*) de gerenciamento de redes. Suporta os padrões SNMPv1, SNMPv2c e SNMPv3. O *Simple Network Management Protocol* (SNMP) é um dentre vários tipos de protocolos que definem mensagens relacionadas ao gerenciamento de redes [ADVENTNET 01].

O SNMP é um protocolo primário, utilizado para transferir dados entre duas ou mais entidades de rede ou nodos. Conforme já comentado no primeiro capítulo deste trabalho, o gerenciamento de redes deve garantir a disponibilidade dos serviços oferecidos pela infraestrutura de rede e para tanto os dispositivos que compõem tal infraestrutura devem ser monitorados e controlados. Gerenciamento local e gerenciamento remoto são dois modos de gerenciamento de um dispositivo conectado à rede. O gerenciamento local requer um gerente humano no local onde o objeto gerenciado está situado. Claramente, redes grandes inviabilizam a aplicação do gerenciamento local. E remotamente, o gerenciamento é facilitado com o SNMP.

Através do SNMP, uma estação rodando uma ou mais aplicações de gerência podem monitorar dados de gerenciamento coletados por vários dispositivos de rede. Tais dados podem ser usados para restabelecer o funcionamento da rede e prever o surgimento de problemas.

5.1 Diferentes Versões do SNMP

A *Internet Engineering Task Force* (IETF) publica documentos denominados *Requests For Comments* (RFCs)[31]. Tais documentos são versões finais especificações de padrões, práticas operacionais, opiniões e outros sobre os protocolos Internet.

Atualmente existem três versões do SNMP:

[31] *http://www.rfc-editor.org/rfcfaq*

- SNMPv1, sendo a primeira versão do protocolo, o qual é definido nas RFCs 1155 e 1157;

- SNMPv2c, é a versão revisada do protocolo, adicionando melhoramentos ao SNMPv1 nas áreas de tipo de pacote, mapeamento de transporte, elementos da estrutura da MIB. Está definido nas RFCs 1901, 1905 e 1906;

- SNMPv3, é a versão segura do SNMP. SNMPv3 também facilita configuração remota de entidades SNMP. Está definido nas RFCs 1905, 1906, 2261, 2262, 2263, 2264 e 2265.

SNMPv1 foi o padrão e primeira versão SNMP. O SNMPv2 foi disponibilizado como uma atualização do SNMPv1 através da adição de novas características. Os melhoramentos chave do SNMPv2 estão concentrados no SMI (discutida nas seções posteriores desse texto), capacidade gerente-para-gerente e operações de protocolo. O SNMPv2c combina a abordagem baseada em comunidade do SNMPv1 com a operação de protocolo do SNMPv2, omitindo todas as características de segurança do SNMPv2. Uma notável deficiência do SNMP era a dificuldade de monitoramento de redes, em oposição de nodos de uma rede. Assim, uma melhora substancial de funcionalidade no SNMP foi a definição de um conjunto de objetos gerenciados padronizados a MIB de monitoramento remoto de rede (*Remote Network Monitoring (RMON) MIB*). Outra grande deficiência era a lucuna de facilidade de segurança. O desenvolvimento do SNMPv3 foi embasado em serviços de segurança como autenticação, privacidade, autorização e controle de acesso. O SNMPv3 define duas capacidades relacionadas com segurança: *User-Based Security Model* (USM) e o *View-Based Security Model* (VACM).

A distribuição AdventNet SNMPv3 suporta as versões v1, v2c e v3 do SNMP [ADVENTNET 01].

5.2 Panorama Histórico

A ARPANET foi a primeira rede de pesquisa financiada pelo DoD (Departamento de Defesa dos EUA), que conectava várias universidades e órgãos governamentais através de linhas telefônicas. Quando redes de rádio e satélite foram introduzidas, os problemas de intercomunicação cresceram e o TCP/IP foi desenvolvido para facilitar a conexão entre essas múltiplas redes. Em primeiro de janeiro de 1983, o TCP/IP tornou-se o único protocolo oficial usado pelo DoD, marcando o declínio da ARPANET. Nascia o embrião da rede hoje denominada Internet.

Apesar do rápido crescimento da Internet na década de 80, não haviam modelos padronizados para gerenciamento. Para preencher a lacuna, três modelos foram desenvolvidos: o *High-Level Entity Management System* (HEMS), o *Common Management Information Protocol* (CMIP) proposto pelo *Open Systems Interconnection* (OSI), um grupo da *International Standards Organization* (ISO) e o *Simple Gateway Monitoring Protocol* (SGMP). O CMIP operando sobre o protocolo TCP foi indicado pela ISO como protocolo de gerenciamento efetivamente utilizado na Internet. Em fevereiro de 1988, o então IAB reuniu um comitê *ad hoc* para determinar qual modelo seria adotado. Optou-se pelo CMOT como uma escolha natural. O SGMP era a solução de curto prazo, utilizada antes do CMOT ser distribuído.

Para uma transição efetiva dos sistemas do SGMP para o CMOT, um *framework* comum para gerenciamento de redes foi desenvolvido para que pudesse ser usado em ambos os modelos. Tal *framework* foi denominado *Simple Network Management Protocol* (SNMP). Em abril de 1989, SNMP recebeu o status de recomendação pelo IAB, como sendo o *framework* de fato em utilização para o gerenciamento de redes. CMOT e SNMP passaram a desenvolver-se de maneira independente. Como tentativa de consenso, em maio de 1990 o IAB tornou o SNMP o protocolo padrão para gerenciamento de redes e um *framework* recomendado para uso na Internet e em todas as redes TCP/IP.

5.3 Vantagens Propiciadas pelo SNMP

Como o nome sugere, o protocolo SNMP é simples de ser entendido e o agente requer apenas software mínimo. Essa simplicidade serve como razão primordial para sua aceitação como padrão de gerenciamento na Internet.

Alguns dos benefícios advindos do emprego do SNMP são:

- padronização. Uma vez que o SNMP é um protocolo de gerenciamento de rede padrão para redes TCP/IP. Protocolos Internet são padrões abertos e não-proprietários, desenvolvidos através dos esforços da comunidade Internet, sofrendo uma ativa manutenção e atualização;

- aceitação abrangente. Todos os maiores fabricantes suportam SNMP. Todos os dispositivos gerenciados pelo SNMP utilizam a mesma interface e suportam um conjunto comum de informações de gerenciamento de rede;

- portabilidade. O SNMP é implementado inteiramente em nível de software e é independente de sistemas operacionais e linguagens de programação. O projeto funcional do SNMP também é portável e define um conjunto básico de operações que devem funcionar de maneira idêntica em todos os dispositivos que suportam SNMP;

- leveza. SNMP facilita a adição de capacidade de gerenciamento para um dispositivo sem um impacto significativo na operação ou desempenho do dispositivo;

- estensibilidade. SNMP é um conjunto básico de operações que se mantém em todos os dispositivos gerenciados. Suporta qualquer tipo de informação em qualquer tipo de dispositivo que seja parte de uma rede de computadores;

5.4 Componentes básicos do SNMP

Os três maiores componentes do SNMP são o dispositivo de rede, o agente e o gerente. Na fig. 5.1 é apresentado o modelo de gerenciamento de redes SNMP.

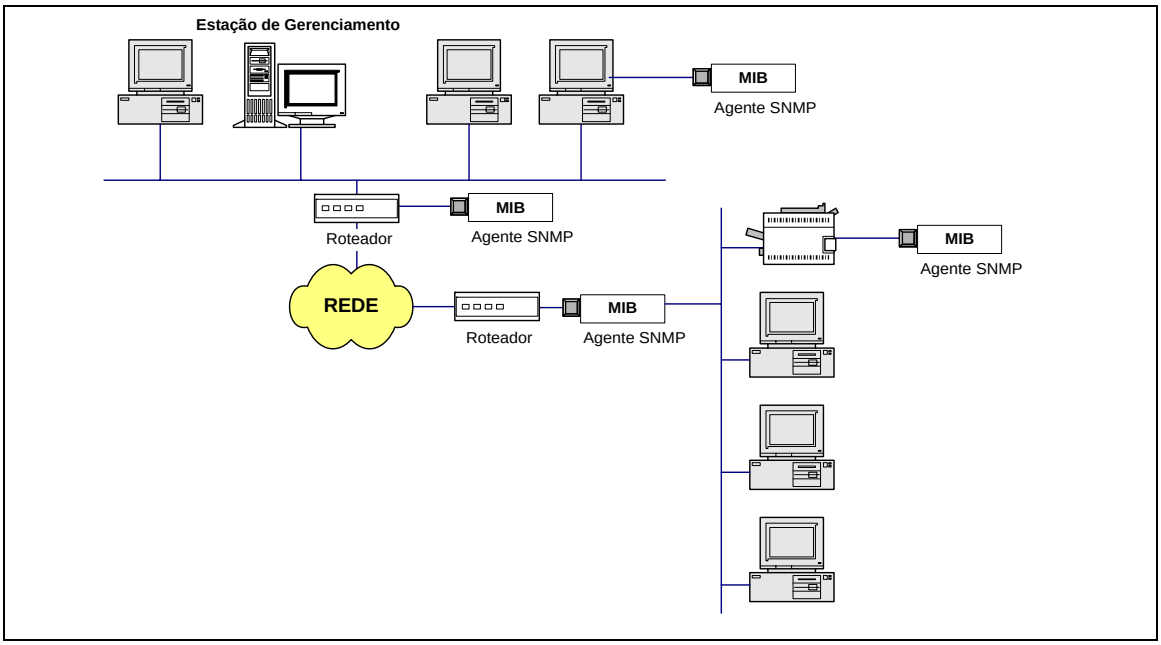

FIGURA 5.1 – COMPONENTES BÁSICOS DO MODELO SNMP

5.4.1 Dispositivo de Rede

O dispositivo de rede ou objeto gerenciado é parte de uma rede que requer algum tipo de monitoramento ou gerência. O agente reside no objeto gerenciado para coletar informação e disponibilizá-la para o gerente.

5.4.2 Agente

Um agente é um mediador entre o gerente e o dispositivo. O agente reside no dispositivo e torna a informação de gerenciamento disponível para o gerente. Entende-se que o agente é um programa que reside no dispositivo ou aplicação e não constitui uma entidade física distinta. Um agente típico deve:

- implementar o protocolo SNMP;
- armazenar e recuperar informações de gerenciamento conforme definido na MIB;

- coletar e manter informação sobre o ambiente local;
- poder enviar de maneira assíncrona um evento para o gerente e
- funcionar como um *proxy* para nodos de rede gerenciados não-SNMP.

5.4.3 Gerente

Um gerente é uma entidade distinta que gerencia os agentes de pontos remotos da rede. Tipicamente é um computador que é utilizado para executar um ou mais sistemas de gerenciamento de rede. Considere, por exemplo, uma organização que possua escritórios em diferentes localizações geográficas. A administração de todos os computadores presentes nas diferentes localidades pode tornar-se difícil. Mas, quando o Sistema Administrador possui um gerente e todos os outros dispositivos e sistemas possuem agentes, a tarefa de gerenciamento é facilitada. O administrador deve apenas consultar o agente através do gerente para estabelecer o estado de funcionamento de um dispositivo. Um gerente típico:

- é implementado como um Sistema de Gerenciamento de Rede;
- deve implementar o protocolo SNMP e
- estar apto para consultar agentes, obter respostas, modificar variáveis nos agentes além de reconhecer e tratar eventos assíncronos gerados pelos agentes.

5.4.4 Comunicação entre o Gerente e o Agente

A comunicação entre o gerente e o agente na rede é implementada através de troca de *Protocol Data Units (PDU)*. Essas PDUs permitem o gerente SNMP interagir com o agente SNMP residente no dispositivo e estender as possibilidades de gerenciamento dependendo daquilo que o agente torna disponível. O protocolo SNMP funciona basicamente em um *processo de "polling"*, onde o gerente consulta periodicamente os

agentes para determinar o estado dos objetos gerenciados. São exceções a esta regra as mensagens de *trap* que são geradas pelo agente. Considerando *traps*, quando uma interface de um roteador falha, por exemplo, um *trap* é enviado ao sistema de gerenciamento notificando o problema. Este poderá então pesquisar o dispositivo em questão para determinar a extensão do problema.

Antes que os dados possam ser transportados através da rede, eles devem ser encapsulados. As PDUs são encapsuladas em *User Datagram Protocol (UDP).* O UDP é um protocolo de transporte não orientado a conexão, incluído no conjunto de procolos TCP/IP e descrito na RFC 768.

Além de operar sob o protocolo de transporte UDP, que é não orientado a conexão, o próprio SNMP é um protocolo não orientado a conexão, sendo cada troca de mensagens entre o gerente e o agente uma transação distinta.

O SNMP é implementado usando abordagem cliente/servidor assíncrona. Assim uma entidade SNMP (estação de gerenciamento ou dispositivo gerenciado) não necessita aguardar por uma resposta depois de enviar uma mensagem. Ele pode enviar outra mensagem se necessário, ou continuar com suas funções predefinidas. Em vez de usar um protocolo baseado no reconhecimento, que consome largura de banda, o SNMP usa a própria resposta como parte de reconhecimento da solicitação.

Cada estação de gerenciamento, bem como os agentes, devem implementar os protocolos SNMP e em razão disso, o UDP e IP, para estabelecer a comunicação. Essa exigência limita o uso do SNMP a dispositivos que implementam totalmente o TCP/IP. Para permitir que dispositivos que não suportam o TCP/IP possam ser gerenciados criou-se o conceito de agente *proxy*. Tal agente é capaz de comunicar-se usando SNMP, em nome de outro dispositivo, retornando o resultado da comunicação ao dispositivo representado de acordo com o protocolo que ele entende. Assim o agente *proxy* estabelece uma função de mapeamento, recebendo as informações do dispositivo representado e convertendo-as em mensagens SNMP e vice-versa.

Na fig. 5.2 pode-se ver a comunicação, através de um agente *proxy*, entre uma estação de gerenciamento e um dispositivo que não suporta SNMP.

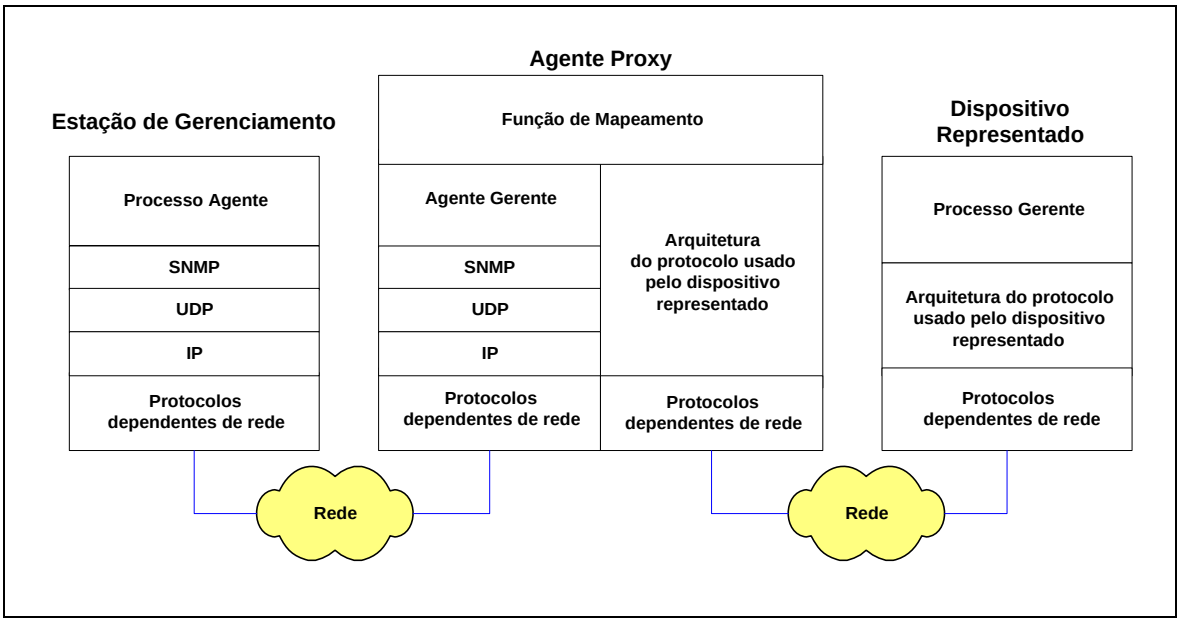

FIGURA 5.2 – CONFIGURAÇÃO DE UM AGENTE *PROXY*

5.4.5 Estrutura

O gerenciamento de rede SNMP é composto por três partes, as quais tanto o gerente quanto os agentes devem estar em conformidade. Tais partes são:

- Protocolo *SNMP*, que define o funcionamento das operações básicas do SNMP e o formato das mensagens trocadas entre o gerente e os agentes.
- *Structure of Management Information (SMI)*, um conjunto de regras usadas para especificar o formato usado para definir objetos gerenciados ou dispositivos que são acessados através do protocolo SNMP. A SMI está definida na RFC 1155 [ROSE 90].
- *Management Information Base (MIB)* é uma coleção de definições, as quais especificam as propriedades do objeto gerenciado.

5.4.5.1 Management Information Base (MIB)

Management Information Bases (MIBs) são coleções de objetos ou definições que especificam as propriedades dos objetos gerenciados. É um banco de dados ativo, possibilitando que suas variáveis sejam recuperadas e atualizadas.

Para permitir o gerente SNMP ou a aplicação de gerência operar inteligentemente os dados disponíveis no dispositivo, o gerente precisa conhecer o nome e os tipos dos objetos no dispositivo gerenciado. Isto se torna possível através dos módulos MIB, que são especificados nos arquivos MIB geralmente fornecidos com os dispositivos gerenciados. A MIB em si é somente uma abstração dos dados.Os padrões de gerenciamento OSI e Internet definiram MIBs que representam os objetos necessários para a gerência de seus recursos.

5.4.5.1.1 MIB OSI

O padrão OSI define três modelos distintos para gerência de redes: o modelo organizacional, o modelo informacional e o modelo funcional. O modelo organizacional descreve a forma pela qual a gerência pode ser distribuída entre domínios e sistemas dentro de um mesmo domínio. O modelo funcional descreve as áreas funcionais e seus respectivos relacionamentos.Já o modelo informacional provê a base para a definição de objetos gerenciados e suas relações, classes atributos, ações e nomes.

Na definição de objetos gerenciados é utilizada a orientação a objetos (OO). Segundo o paradigma OO, objetos com características semelhantes agrupam-se em classes de objetos. Uma classe, dita derivada ou filha, pode ser uma subclasse de outra chamada classe base ou superclasse. A classe filha herda todas as propriedades da classe base. Uma classe é definida pelos atributos da classe, pelas ações que podem ser invocadas, pelos eventos que podem ser relatados, pela subclasse a qual ela deriva e pela superclasse na qual ela está contida.

Para a definição dos objetos gerenciados deve-se considerar três hierarquias:

- **Hierarquia de Herança –** Também denominada hierarquia de classe, tem como objetivo facilitar a modelagem dos objetos, através da utilização do

paradigma da orientação a objetos. Assim podem ser definidas classes, superclasses, subclasses. Trata-se de uma ferramenta para uma melhor definição de classes.

- **Hierarquia de Nomeação** – A hierarquia de nomeação, também chamada hierarquia de *containment*, descreve a relação de "estar contido em" aplicado aos objetos. Um objeto gerenciado está contido dentro de um e somente um objeto gerenciado. Um objeto gerenciado existe somente se o objeto que o contém existir, e dependendo da definição, um objeto só pode ser removido se aqueles que lhe pertencerem forem removidos primeiro.

- **Hierarquia de Registro** – A hierarquia de registro é usada para identificar de maneira universal os objetos, independentemente das hierarquias de heranças e nomeação. Esta hierarquia é expressa segundo regras estabelecidas pela notação de dados baseada em texto denominada *Abstract Syntax Notation One* (*ASN.1*), uma descrição de dados não ambígua no formato de texto ASCII. Assim, cada objeto é identificado por uma seqüência de números, correspondente aos nós percorridos desde a raiz, até o objeto em questão. Esta hierarquia é também usada pelo padrão Internet.

5.4.5.1.2 MIB Internet

A primeira versão da MIB destinada ao protocolo TCP/IP foi definida na RFC 1066 e ficou conhecida como MIB-I. A RFC 1066 definiu a base de informação necessária para monitorar e controlar redes baseadas no protocolo TCP/IP. O RFC 1066 foi aceito pelo IAB (*Internet Activities Board*) como padrão na RFC 1156.

A MIB-II é um superconjunto da MIB-I com alguns objetos e grupos adicionais, estando definida na RFC 1213.

As MIBs padrão são todas aquelas aprovadas pelo IAB. Fabricantes de software e equipamentos definem MIBs privadas de maneira unilateral. A distinção entre MIBs padrão e privada está baseada em como as variáveis estão definidas.

O melhor exemplo de uma MIB padrão é a rfc1213-MIB (também conhecida como MIB-II). É um modulo MIB tipicamente suportado por todos os agentes SNMP em dispositivos TCP/IP disponíveis.

No padrão Internet os objetos gerenciados são definidos em uma árvore de registro, equivalente a hierarquia de registro do padrão OSI. O módulo MIB contém a descrição de uma hierarquia de objetos nos dispositivos gerenciados, bem como o nome (*Object ID*), sintaxe e privilégios de acesso para cada variável na MIB.

Por exemplo:

```
directory(1)
identificador de objetos: 1.3.6.1.1
descrição textual: {internet 1}
```

Um nó rotulado pode possuir subárvores, que por sua vez podem conter outros nós rotulados. Caso não tenha subárvores, ou nós folhas, o nó rotulado conterá um valor e será um objeto.

O nó raiz da árvore MIB não possui nome ou número, mas tem três sub-árvores:

- ccitt(0), administrada pelo CCITT;
- iso(1), administrada pela ISO;
- joint-iso-ccitt(2), administrada pela Isso juntamente com o CCITT.

Sob o nó iso(1), estão outras sub-árvores, como é o caso da sub-árvore org(3), definida pela ISO para conter outras organizações. Uma das organizações que está sob a sub-árvore org(3) é o DoD (Departamento de Defesa dos EUA), no nó dod(6). A Internet(1) está sob o dod(6), possuindo quatro subárvores:

- directory(1): contém informações sobre o serviço de diretórios OSI (X.500);

- mgmt(2): contém informações de gerenciamento, é sob esta subárvore que está o nó da mibII, com o identificador de objeto 1.3.6.1.2.1 ou { mgmt 1 };

- experimental(3): contém os objetos que ainda estão sendo pesquisados pelo IAB;

- private(4): contém objetos definidos por outras organizações.

A fig. 5.3 apresenta parte da estrutura em árvore de uma MIB.

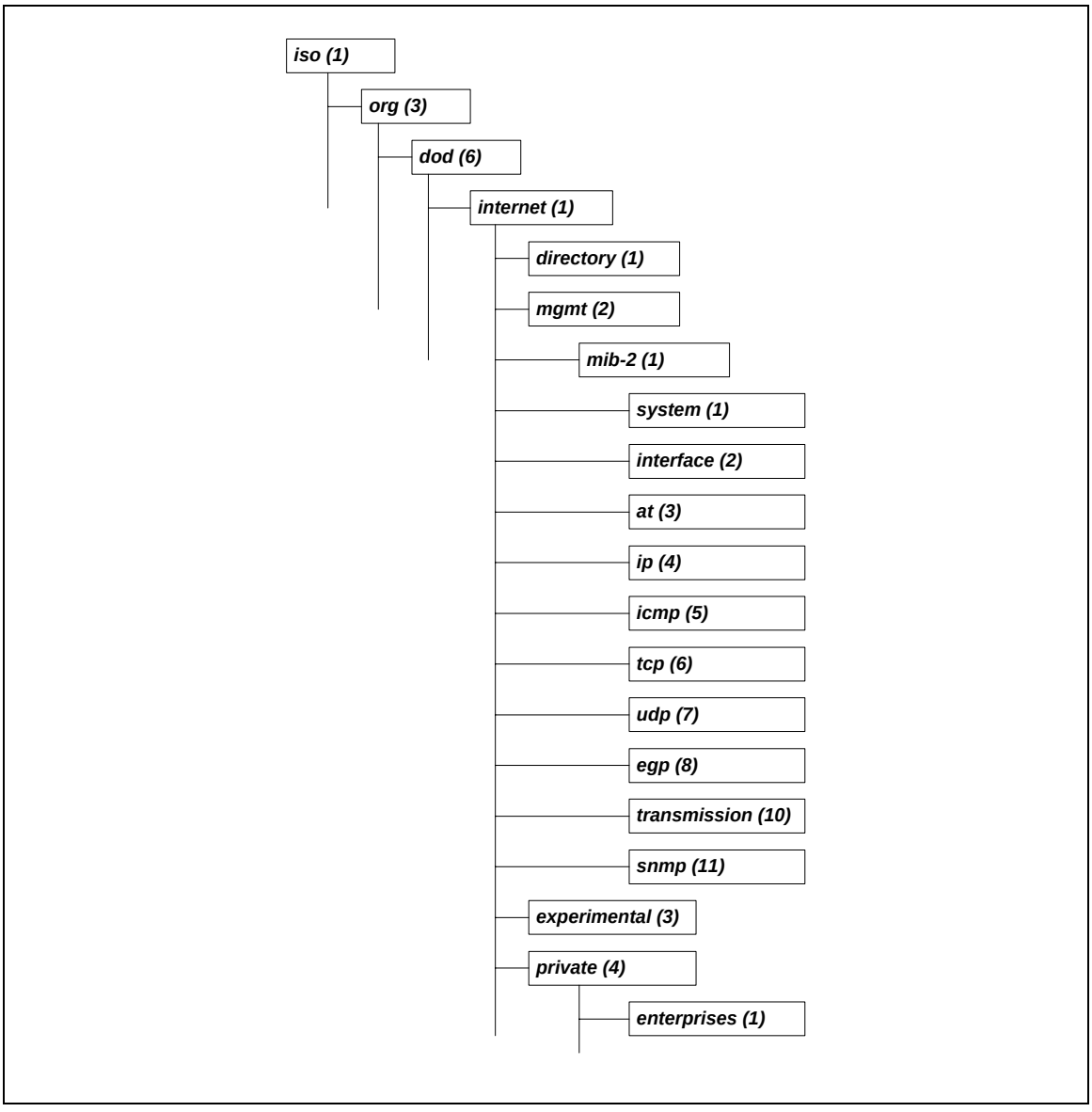

FIGURA 5.3 – FRAGMENTO DA ESTRUTURA DE UMA MIB

Abaixo da subárvore mibII estão os objetos usados para obter informações específicas dos dispositivos da rede. Esses objetos são categorizados em 11 grupos, que são apresentados na tabela 5.1.

TABELA 5.1 – GRUPOS DA MIB-II

Grupos	**Informações**
System(1)	Sistema de operação dos dispositivos da rede
Interfaces(2)	Interface da rede com o meio físico
Address translation(3)	Mapeamento de endereços IP em endereços físicos
Ip(4)	Protocolo IP
Icmp(5)	Protocolo ICMP
Tcp(6)	Protocolo TCP
Udp(7)	Protocolo UDP
Egp(8)	Protocolo EGP
Cmot(9)	Protocolo CMOT
Transmission(10)	Meios de transmissão
Snmp(11)	Protocolo SNMP

Cada objeto contido nos grupos apresentados na tabela 1 é descrito no RFC1213. A descrição dos objetos é dividida em cinco partes: o nome do objeto, a sintaxe abstrata do objeto, a descrição textual do significado do objeto, o tipo de acesso permitido ao objeto (*read-only*, *read-write*, *write-only* ou não acessível), e o estado do objeto (obrigatório, opcional, obsoleto).

Um aspecto chave das MIBs é que somente o tipo dos objetos no dispositivo gerenciado está definido na MIB e não instâncias específicas. Por exemplo, ifInOctets na rfc1213-MIB especifica um tipo de objeto, para número de octetos de entrada em uma interface, mas as instâncias específicas desse tipo são especificadas como ifInOctets.1, ifInOctets.2, etc., dependendo do número de interfaces. Quando especificar um objeto para o agente SNMP, um Object ID apropriado, que inclui a instância,

precisa ser informado pelo gerente. Quando não especificado corretamente, o agente responde com um erro "*No such variable*".

5.4.5.2 OID

O gerente SNMP ou aplicações de gerenciamento, utilizam nomes com sintaxe bem definida para especificar as variáveis de interesse para o agente SNMP. Nomes de objeto nessa sintaxe são denominados *Object Identifiers* (*object Ids* ou OIDs), e são uma série de números que univocamente identificam um objeto para o agente SNMP.

OIDs são arranjados em uma estrutura hieráquica, de árvore invertida. A árvore OID inicia com a raiz e expande-se abaixo para os ramos. Cada ponto na árvore OID é chamado de nodo e cada nodo pode possuir um ou mais ramos, ou pode terminar com um nodo folha. O formato do OID é uma seqüência de números separados por pontos. Existem duas raízes para identificadores de objetos: iso (.1) e ccit (iniciando com .0) . Muitos idenficadores iniciam com .1.3.6.1 (onde 1=iso, 3=org, 6= dod, 1 = internet). Os ramos internet dividem-se em mgmt e private. Todas as MIB padrão estão sob *mgmt*, enquanto que as MIBs privadas estão abaixo de *private.enterprises*.

Para esclarecer o conceito de identificadores de objeto Relativo e Absoluto, consideremos o identificador de objeto AdventNet .1.3.6.1.4.1.2162. Ele especifica um caminho a partir da raiz da árvore. A raiz não tem um nome ou um número, mas o 1 inicial neste OID está diretamente abaixo da raiz. Esse é chamado de OID absoluto. Por outro lado, caminhos para variável podem ser especificados relativamente a algum nodo na árvore OID. Por exemplo, 2.1.1.7 especifica o objeto *sysContact* no grupo *system*, relativo ao nodo *internet* (.1.3.6.1) na árvore OID. Este é um OID relativo.

Para obter valores de objetos do agente, é necessário especificar a instância do objeto. Adicionando o índice de instância ao OID especifica-se a instância do mesmo. Por exemplo, o último 0 em *.iso.3.dod.1.mgmt.mib.1.sysUpTime.0*, é o índice de instância. Um índice de intância “0” indica a primeira intância, “1” a segunda, e assim sucessivamente. Sendo *sysUpTime* um objeto escalar, ele possui apenas uma única instância. Então, um índice de instância de 0 é sempre especificado quando recupera-se

o valor de um objeto escalar. Um índice de instância maior do que 0 pode somente ser utilizado no caso de objetos colunares (em tabela), que podem possuir múltiplas instâncias.

5.4.5.3 SMI e versões SMI

Structure of Management Information é um conjunto de regras usadas para especificar o formato usado na definição de objetos gerenciados ou dispositivos acessados através do SNMP.

A SMI descreve a árvore de nomes MIB, que é usada para identificar objetos gerenciados e definir os ramos da árvore MIB onde residem os objetos SNMP gerenciados.

As duas versões do SMI são SMIv1 e SMIv2. SMIv1 é definido na RFC 1155, RFC 1212 e RFC 1215 e o SMIv2 é definido nas RFCs 1902, 103 e 1904.

SMIv2 é retroativamente compatível com o SMIv1. Isso significa que é possível converter uma MIB SMIv2 para SMIv1, com exceção daqueles objetos do tipo de dados Counter64. Porém, a conversão de SMIv1 para SMIv2, não é automática. Isso se deve ao fato do SMIv2 englobar o SMIv1. Também, o formato SMIv2 contém construções para definir requisição de especificações e implementação de requisições, que não integram o formato SNMPv1.

5.4.5.4 SMI Data Types

Tipos de dado podem ser inteiro, ponto flutuante, string octeto e identificador único. Os módulos MIB e SMI são expressos em ASN.1. Os dados da MIB são transportados pela rede usando mensagens SNMP, as quais são codificadas em *Basic Encoding Rules* (*BER*). São similares ao SMI, mas as mensagens são codificadas no formato binário. Tanto o ASN.1 e BER são essenciais para a implementação do SNMP.

Os tipos disponibilizados pelo ASN.1 são categorizado em tipos simples e tipos estruturados. Os desenvolvedores do SNMP escolheram os tipos simples, os quais incluem os tipos INTEGER, OCTET STRING e OBJECT IDENTIFIER. Os outros tipos escolhidos foram *Gauge*, *Counter*, *TimeTicks*, *IPAddress*, *NetworkAddress* e *Opaque*.

A API SNMP suporta os padrões SMIv1 e SMIv2. Os vários tipos de dados para informação de gerenciamento são apresentados na Tabela 5.1.

TABELA 5.2 – TIPOS DE DADOS SMI

Tipos de Dados SMIv1	**Tipos de Dados SMIv2**
INTEGER	Integer32
INTEGER(enumerated integer)	INTEGER(enumerated integer) Unsigned32
Gauge	Gauge32
Counter	Counter32 Counter64
TimeTicks	TimeTicks
OCTET STRING	OCTET STRING
OBJECT IDENTIFIER	OBJECT IDENTIFIER
IpAddress	IpAddress
NetworkAddress	
Opaque	Opaque
	BITS

5.4.5.5 Tipos de Dados MIB SMIv1

O SMIv1 define os seguintes tipos de dados:

- INTEGER. Usado para especificar valores nas quais a faixa inclui números positivos e negativos. O SNMIv1 não especifica valores mínimos ou máximos para a faixa;

- ENUMERATED INTEGER. Usado para especificar uma lista de valores inteiros nomeados. No SNMIv1 os valores devem ser maiores que zero, enquanto que no SNMIv2 permite qualquer valor na faixa 2^{31} a 2^{31}-1;

- GAUGE. Representa um inteiro não-negativo que pode ser incrementado ou decrementado, mas até um certo valor máximo ou mínimo, respectivamente;

- COUNTER. Usado para especificar um valor que representa um contagem. A faixa é 0 até 4294967295;

- TIMETICKS. Usado para especificar o tempo decorrido entre dois eventos, em unidades de milésimos de segundo. A faixa é 0 até 2^{32}-1;

- OCTET STRING. Usado para especificar octetos binários ou informação textual. Enquanto o SMIv1 não limita o número de octetos, o SMIv2 determina um limite de 65535 octetos. Um tamanho pode ser especificado como sendo fixo, variante ou de múltiplas faixas;

- OBJECT IDENTIFIER. Usado para identificar um tipo que tenha um atribuído um valor de identificador de objeto;

- IPADDRESS. Usado para especificar um endereço Ipv4 como uma string de 4 octetos;

- NETWORKADDRESS. Obsoleto. O SMIv2 suporta esse tipo através do tipo IPADDRESS;

- OPAQUE. Usado para especificar octetos e informação binária. O SMIv2 determina um limite de 65535 octetos, enquanto o SMIv1 não impõe limitação. Um valor desse tipo deve um encapsulamento de um valor ASN.1 BER codificado.

5.4.5.6 Tipos de Dados MIB SMIv2

O SMIv2 define os seguintes tipos de dados:

- INTEGER32. Usado para especificar valores nas quais a faixa inclui números positivos e negativos. O SNMIv2 não especifica valores mínimos ou máximos para a faixa;

- ENUMERATED INTEGER. Usado para especificar uma lista de valores inteiros nomeados. No SNMIv2 são permitidos quaisquer valores pertencentes a faixa 2^{31} a 2^{31}-1, enquanto que no SNMIv1 os valores devem ser maiores que zero;

- UNSIGNED32. Usado para especificar valores não negativos na faixa 0 até a 2^{31}-1;

- GAUGE32. Representa um inteiro não-negativo que pode ser incrementado ou decrementado, mas até um certo valor máximo ou mínimo, respectivamente;

- COUNTER32. Usado para especificar um valor que representa um contagem. A faixa é 0 até 4294967295;

- COUNTER64. Similar ao COUNTER32, com exceção da faixa que vai de 0 a 2^{64}-1. Como esse tipo não existe no SMIv1, deve ser empregado apenas quando compatibilidade retroativa não é requerida;

- TIMETICKS. Usado para especificar o tempo decorrido entre dois eventos, em unidades de milésimos de segundo. A faixa é 0 até 2^{32}-1;

- OCTET STRING. Usado para especificar octetos binários ou informação textual. O SMIv2 determina um limite de 65535 octetos, ao contrário do SMIv1 que não limita o número de octetos. Um tamanho pode ser especificado como sendo fixo, variante ou de múltiplas faixas;

- OBJECT IDENTIFIER. Usado para identificar um tipo que tenha um atribuído um valor de identificador de objeto;
- IPADDRESS. Usado para especificar um endereço Ipv4 como uma string de 4 octetos;
- NETWORKADDRESS. Obsoleto. O SMIv2 suporta esse tipo através do tipo IPADDRESS.
- OPAQUE. Usado para especificar octetos e informação binária. O SMIv2 determina um limite de 65535 octetos, enquanto o SMIv1 não impõe limitação. Um valor desse tipo deve um encapsulamento de um valor ASN.1 BER codificado;
- BITS. Usado para especificar uma coleção nomeada de bits. Provê um modo de identificar os bits individuais em um octeto (uma extensão do tipo OCTET STRING).

5.4.6 Operações SNMP Básicas

Conforme já comentado, o SNMP é um protocolo solicitação-resposta. As operações realizadas são categorizadas como :

- Recuperação de dados - GET, GET NEXT e GET BULK;
- Alteração de variáveis – SET e
- Recebimento de mensagens não solicitadas – *TRAP*S.

5.4.6.1 Recuperação de Dados

Uma requisição é enviada pelo gerente para um agente a fim de retornar dados através da execução de uma operação GET, GETNEXT ou GETBULK.

A operação GET é uma requisição enviada do gerente para o objeto gerenciado através do agente. Quando executada retorna um ou mais valores do objeto gerenciado. Já a operação GETNEXT é similar a operação GET. A diferença significante é que o GETNEXT retorna o valor do próximo OID na árvore. Finalmente, GETBULK retorna grandes volumes de dados.

5.4.6.2 Alteração de Variáveis

O gerente pode em dado momento alterar variáveis. Essa funcionalidade é provida pela operação SET.

5.4.6.3 Recebimento de Mensagens Não Solicitadas

Quando o agente identifica um erro na transmissão da mensagem, responde ao gerente enviando uma mensagem não solicitada através da operação *TRAP*.

5.4.7 Formato das Mensagens SNMP

No protocolo SNMP a informação é trocada entre o gerente e um agente na forma de mensagem. Cada mensagem inclui um numero de versão, indicando a versão do SNMP, um nome de comunidade e um dos cinco tipos de PDU: *GetRequest*, *GetNextRequest*, *GetResponse*, *SetRequest* e *Trap*. O formado de cada PDU é mostrado na fig. 5.4 e a descrição dos campos que a constituem são apresentados na Tabela 5.2 [STALLINGS 00].

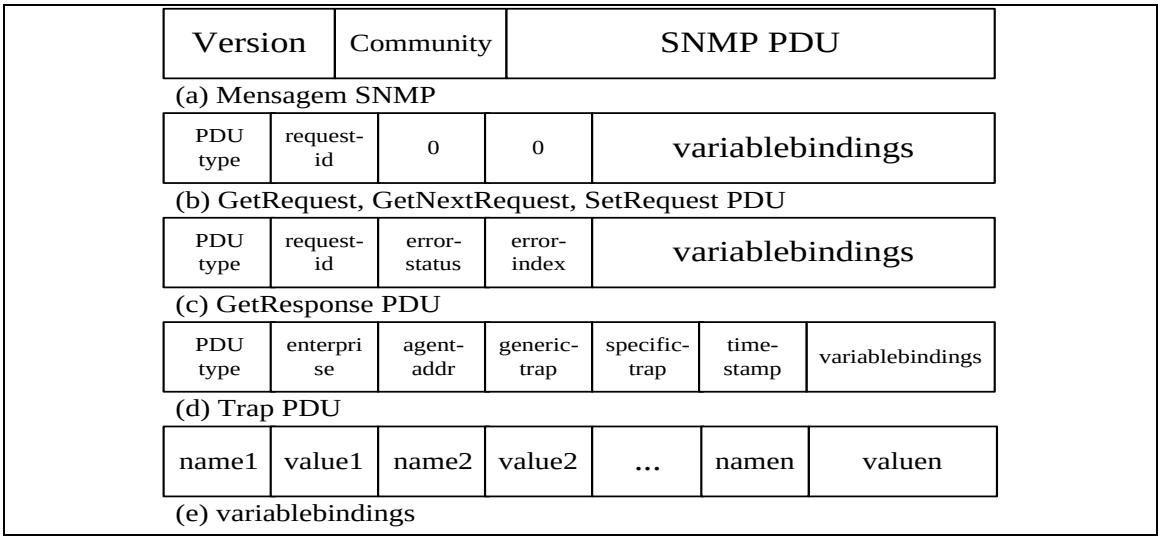

FIGURA 5.4 – FORMATO DAS PDUS SNMP

As PDU *GetRequest*, *GetNextRequest* e *SetRequest* têm o mesmo formato da PDU *GetResponse* com os campos *error-status* e *error-index* sempre tendo o valor zero. Esta convenção reduz para um o número de diferentes formatos de PDU com que a entidade SNMP deve lidar [STALLINGS 98].

TABELA 5.3 – DESCRIÇÃO DOS CAMPOS DA PDU SNMP

Campo	**Descrição**
version	Versão do SNMP.
community	Nome da comunidade, agindo como uma forma de autenticação.
request-id	Usado para distinguir as mensagens enviadas provendo para cada requisição um único ID.
error-status	Indica que uma exceção ocorreu durante o processamento de uma requisição
error-index	No caso de ocorrer um *error-status*, este campo contém uma informação adicional indicando qual variável causou a exceção.
variablebindings	Uma lista de nomes de variáveis e seus respectivos valores.
enterprise	O tipo de objeto gerador da *trap*.
agent-addr	Endereço do objeto gerador da *trap*.

generic-trap	Especifica os tipos genéricos de *trap*.
specific-trap	Código para uma *trap e*specífica.
time-stamp	Tempo decorrido entre a última inicialização da entidade de rede e a geração da *trap*.

As mensagens entre o agente e o gerente, com diferentes identificadores de versão, são simplesmente descartadas sem sofrer qualquer forma de processamento. O nome da comunidade age como um mecanismo simplificado de senha entre os gerentes e agentes, ou seja, quando o agente recebe uma solicitação do gerente, ele confere se os nomes das comunidades são iguais, caso não sejam, retorna um *trap* indicando uma falha de autenticação [BAROTTO 98].

As PDUs *GetRequest* e *GetNextRequest* são comandos do gerente para receber dados de um agente. A diferença é que a operação *GetRequest* lista um valor especifico de uma ou mais variáveis, enquanto que a operação *GetNextRequest* é usada para percorrer a árvore MIB. Em ambos os casos, os valores, se disponíveis, são retornados ao gerente através da PDU *GetResponse*. A operação *SetRequest* é um comando do gerente para o agente, utilizado para atualizar variáveis no agente, sendo que neste caso a PDU *GetResponse* provê uma mensagem de confirmação. Finalmente, a PDU *Trap* é uma notificação do agente para o gerente [STALLINGS 98].

6. Domínio da Aplicação

Conforme exposto no Capítulo 1 (Introdução), um dos objetivos principais desse trabalho é demonstrar uma aplicação de XML na Gerência de Redes.

Para atingir tal objetivo foi implementado um aplicativo de gerenciamento que extrai informações gerenciais de dispositivos de rede dotados de agentes SNMP. O aplicativo comunica-se com o agente SNMP do dispositivo através de consultas SNMP e gerar documentos XML a partir das respostas recebidas. Essa aplicação foi desenvolvida em Java 2. Tais documentos poderão ser processados por ferramentas XML para serem exportados/apresentados em diferentes formatos.

O ambiente de rede que serviu de teste pertence a UNIOESTE (Universidade Estadual do Oeste do Paraná), campus de Cascavel, Paraná. A UNIOESTE é uma das instituições que compõem a Intranet Paraná. Intranet Paraná é a primeira rede de alta velocidade que integra todas as regiões paranaenses. Base para o sistema estadual de Ciência e Tecnologia, objetiva criar as condições ideais para a conexão de um canal internacional de dados com acesso direto à Internet e viabilizar a interoperabilidade entre as redes de informações das instituições de ensino superior e pesquisa. Constitui um *backbone* servindo a 45.000 usuários entre pesquisadores, professores e alunos. Tem como ponto central da rede o Centro de Operações da Intranet Paraná em Curitiba, podendo ser estendida a pontos remotos, dentro e fora do Paraná com *links* exclusivos de satélite [INTRANET 02]. A fig. 6.1 apresenta os pontos de presença da Intranet Paraná.

Um dos fatos que merece atenção especial dos gerentes de rede é o congestionamento. Um acréscimo gradativo da situação de congestionamento, não tratada pelo gerente, pode ocasionar uma paralisação completa da rede. Erros ou mesmo sobrecargas de certos segmentos da rede podem gerar “gargalos”.

Tendo em vista a situação de possíveis gargalos, a aplicação desenvolvida propõe-se a apresentar a situação do tráfego dos *links* TCP/IP que a UNIOESTE mantêm, principalmente com seus outros *campi*, situados nos municípios de Foz do Iguaçu, Francisco Beltrão, Marechal Cândido Rondon e Toledo). Avaliações sobre o tráfego são

de grande valia particularmente em um momento que a instituição passa por um processo de reestruturação de seus sistemas de informação, antes baseados em sistemas isolados em cada campus para sistemas integrados e distribuídos, tendo *browsers Web* como uma ferramenta típica para interface com os usuários.

FIGURA 6.1 – PONTOS DE PRESENÇA DA INTRANET PARANÁ

Para a obtenção das informações referentes ao tráfego, foi considerado um roteador, equipamento do fabricante Cisco, da série Cisco 7000 e fisicamente instalado na Diretoria de Informática, na Reitoria da UNIOESTE, no município de Cascavel,

Paraná. Tal roteador, como mostrado esquematicamente na fig. 6.2, possui uma MIB e um agente SNMP, que dessa forma permite a consulta às variáveis SNMP através de primitivas SNMP.

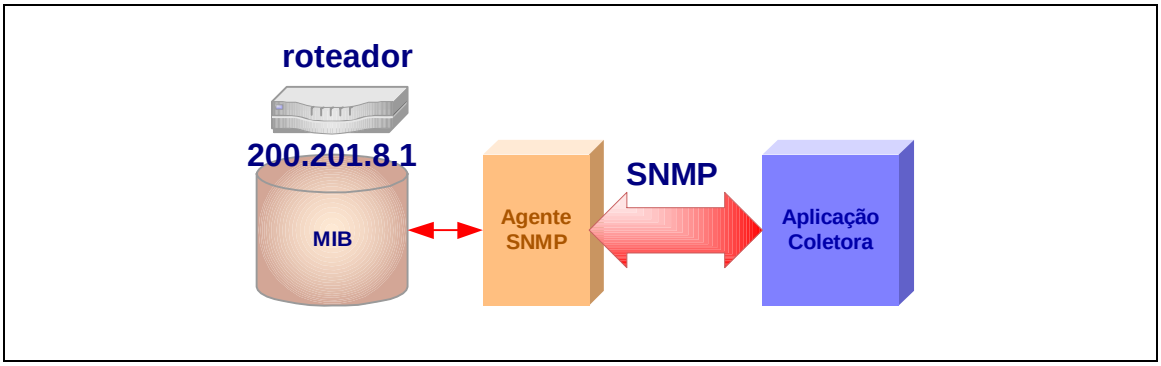

FIGURA 6.2 – ROTEADOR MONITORADO

6.1 Ferramentas Utilizadas

Para a implementação dessa aplicação foram utilizadas as seguintes ferramentas de software:

- pacote Java 2 SDK versão 1.4.0, da Sun Microsystems, que inclui o compilador Java, a máquina virtual Java e a biblioteca de classes padrão da linguagem;

- IDE (*Integrated Development Environment*) destinado a tecnologias Java JCreator 2.00 PRO, da empresa Xinox Software. O software possui interface enxuta e leve, diferente de outras soluções disponíveis no mercado que se caracterizam como exigindo consideráveis recursos da máquina para executar a própria interface. O JCreator possui um gerenciador de projetos, *templates* de código, *browsers* de classes, complementação de código na digitação (*code completion*), depurador integrado a interface, destacamento de sintaxe (*syntax highlighting*), assistentes (*wizards*) para algumas tarefas e uma interface de usuário customizável;

- API AdventNet SNMP, um conjunto de classes implementadas em Java que permitem o envio e recebimento de primitivas SNMP [ADVENTNET 01];

6.2 Variáveis Monitoradas

As variáveis monitoradas pertencem ao grupo *interfaces* da MIB2 – Internet especificada na RFC 1213. O Grupo *Interfaces* oferece dados sobre cada *interface* de um dispositivo gerenciável da rede. Essas informações são úteis para o gerenciamento de falhas, de configuração, de desempenho, e de contabilização. As variáveis monitoradas são:

- **ifOutOctets –** o número total de bytes transmitidos por uma *interfac*e, incluindo caracteres de cabeçalho. Nome: IF-MIB!ifOutOctets; Identificador: 1.3.6.1.2.1.2.2.1.16;

- **ifOutDiscards** – o número de pacotes a serem transmitidos por uma *interface* acima do limite. Estes pacotes são escolhidos para serem descartados mesmo que não tenham sido detectados erros. Nome: IF-MIB!ifOutDiscards; Identificador: 1.3.6.1.2.1.2.2.1.19;

- **ifOutErrors –** o número de unidades de transmissão que contiveram erros. Estes pacotes ou unidades de transmissão são descartados, impedindo que os mesmos se propaguem pela rede. Nome: IF-MIB!ifOutErrors; Identificador: 1.3.6.1.2.1.2.2.1.20;

- **ifInOctets –** o número total de bytes recebidos em uma *interfac*e, incluindo caracteres de cabeçalho. Nome: IF-MIB!ifInOctets; Identificador: 1.3.6.1.2.1.2.2.1.10;

- **ifInDiscards –** o número de pacotes recebidos em uma *interface* acima do limite. Estes pacotes são escolhidos para serem descartados mesmo que não tenham sido detectados erros. Uma razão para descartar pacotes é que

ele pode ser maior do que o espaço livre no *buffer* do roteador. Nome: IF-MIB!ifInDiscards; Identificador: 1.3.6.1.2.1.2.2.1.13;

- **ifInErrors –** o número de unidades de transmissão recebidos com erros. Estes pacotes ou unidades de transmissão são descartados, impedindo que os erros se propaguem para o protocolo de nível mais alto. Nome: IF-MIB!ifInErrors; Identificador: 1.3.6.1.2.1.2.2.1.14;

Na aplicação, além de utilizar o identificador da variável monitorada é necessário adicionar, no final do identificador, o número da porta monitorada. Por exemplo: a constante ifOutOctets tem como identificador o número 1.3.6.1.2.1.2.2.1.16 porém para a coleta de dados, deve-se acrescentar o ".4" para indicar que a porta monitorada é a porta 4 [VERONEZ 99].

6.3 Arquitetura da Aplicação de Coleta de Dados Gerenciais

A arquitetura da aplicação implementada é mostrada na fig. 6.2.

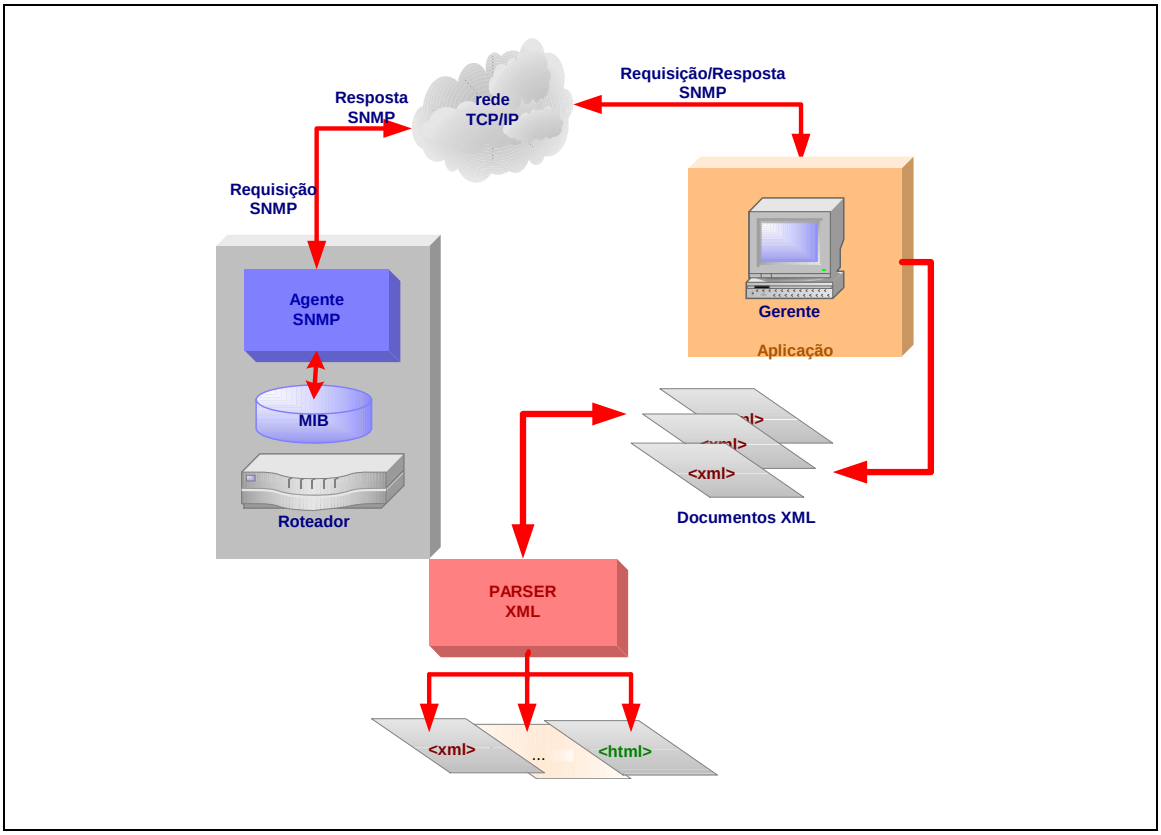

FIGURA 6.3 – ARQUITETURA DA APLICAÇÃO DE COLETA DE DADOS GERENCIAIS

Basicamente, a aplicação utiliza o protocolo SNMP para consultar o roteador (endereço IP 200.201.8.1) onde estão fisicamente conectados os *links* e retornar informações de tráfego mantidos na MIB desse equipamento. A partir das informações recebidas é gerado um documento XML.

Nos testes, as consultas foram realizadas em intervalos regulares de 5 minutos. Como as variáveis monitoradas retornam octetos, é feita uma conversão para uma unidade mais usual, de bits por segundo (*bits/s*). O principal objetivo é prover estatísticas de tráfego ao administrador da rede.

Na fig. 6.4 é apresentada a interface principal da aplicação de coleta.

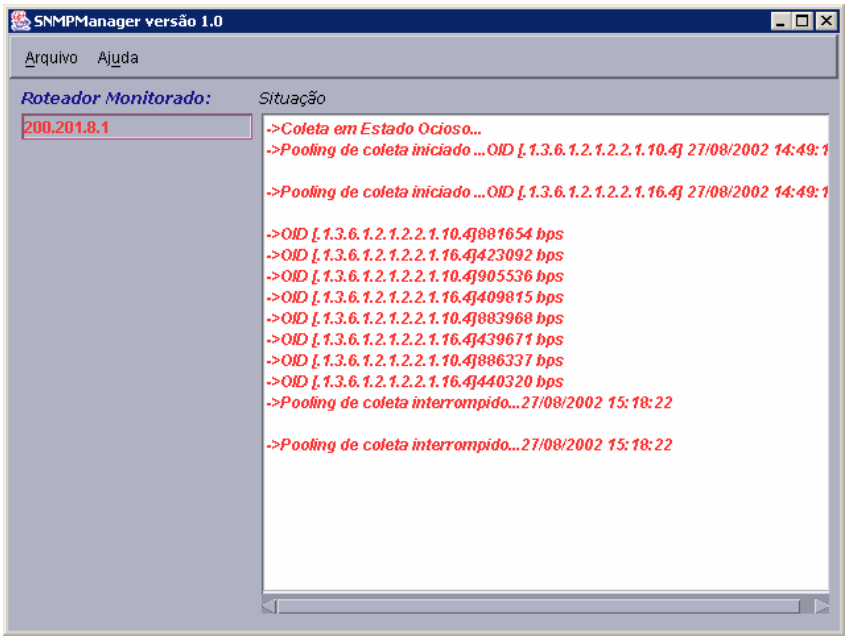

FIGURA 6.4 – INTERFACE PRINCIPAL DA APLICAÇÃO DE COLETA DE DADOS

Basicamente, a aplicação utiliza o protocolo SNMP para consultar o roteador (endereço IP 200.201.8.1) onde estão fisicamente conectados os *links* e retornar informações de tráfego mantidos na MIB desse equipamento. A partir das informações recebidas é gerado um documento XML. Tal documento pode ser reprocessado por aplicações XML e renderizado para produzir novos formatos de saída. Na fig. 6.4 é apresentada a saída XML renderizada para XHTML e o respectivo documento aberto no *browser*.

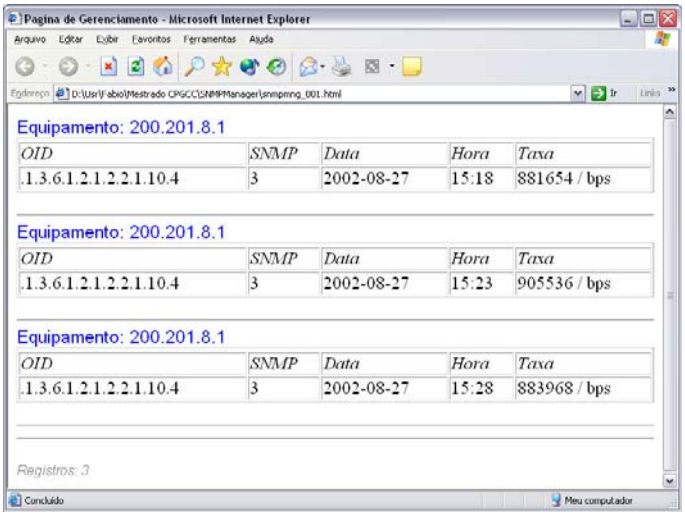

FIGURA 6.5 – SAÍDA XML RENDERIZADA PARA XHTML

6.4 Arquitetura do ambiente hipotético utilizando CORBA

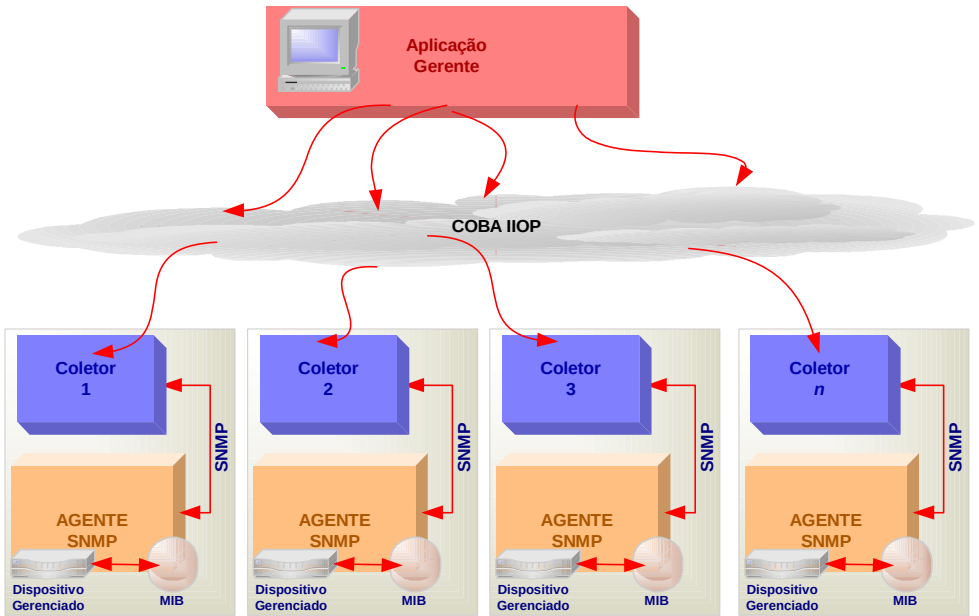

FIGURA 6.6 – MODELO LÓGICO DO AMBIENTE APLICANDO CORBA

7. Conclusões

7.1 Conclusão

Atualmente, as redes de computadores baseadas na arquitetura TCP/IP, notadamente a Internet, crescem assustadoramente. Tal expansão vigorosa concentra-se na relativa facilidade de implementação e manutenção do conjunto de protocolos TCP/IP, além da possibilidade de interligação de LAN's através de outras WAN's, com um desempenho bastante razoável. É possível, inclusive, ter-se uma implementação de rede local utilizando a arquitetura TCP/IP, sem contudo conectá-la a outras redes, caracterizando uma rede intranet, bastante empregada na implementação de LANs em empresas [STALLINGS 00].

Porém, torna-se imprescindível o monitoramento e controle do funcionamento e crescimento dessas redes, assegurando um desempenho aceitável. A necessidade de gerenciamento de redes é evidente e tende a crescer à medida que as redes se tornam maiores e mais complexas. Como essas redes apresentam-se cada vez mais heterogêneas, uma padronização do protocolo de gerência garante que estas sejam gerenciadas de maneira uniforme.

Dentre as atividades básicas do gerenciamento de redes estão a detecção e correção de falhas e o estabelecimento de procedimentos para a previsão de problemas futuros. Por exemplo, monitorando linhas cujo tráfego esteja aumentando ou roteadores que estejam se sobrecarregando, é possível tomar medidas que evitem o colapso da rede, como a reconfiguração das rotas ou a troca do roteador por um modelo mais adequado.

O protocolo de gerência SNMP constitui, atualmente, um padrão operacional, e grande parte do seu sucesso se deve a sua simplicidade. Outro aspecto importante é a sua capacidade de gerenciar redes heterogêneas constituídas de diferentes tecnologias, protocolos e sistemas operacionais. Dessa forma, o SNMP é capaz de gerenciar redes *Ethernet*, *Token Ring* que conectem PC's, *Apple Machintosh*, estações SUN e outros

tipos de computadores [STALLINGS 00]. Jeffrey Case, presidente do SNMP *Research International Inc*., certa vez afirmou que "em um mundo ideal, eu usaria SNMP para coleta de dados, *browsers* para apresentação dos dados, Java para a inteligência e um esquema padronizado para a portabilidade dos dados" [JANDER 96].

O emprego de padrões como o SNMP veio tratar a necessidade de gerenciamento da infra-estrutura de rede de computadores das instituições, através de um modelo aberto, não proprietário, baseado no princípio gerente-agente.

Tecnologias como CORBA começam a ser utilizadas no desenvolvimento dos chamados *frameworks* de forma a tornar disponível um mecanismo transparente para comunicação entre os agentes e gerentes distribuídos na rede além de oferecer uma API normatizada para o desenvolvimento de aplicações distribuídas. Várias empresas têm questionado os altos custos relacionados às plataformas de gerenciamento. Os fabricantes de equipamentos questionam também o esforço de desenvolvimento para portar suas aplicações de gerência para uma miscelânea de plataformas disponíveis no mercado.

Graças à massificação das redes TCP/IP, a Internet constitui uma infraestrutura ideal para prover intercâmbio de informações. Uma das possibilidades proporcionadas pela *Web* é o acesso e tratamento de informações de gerenciamento de redes. Nessa conjuntura, CORBA já vem sendo utilizada sobre a infraestrutura *Web* [BARILLAUD 97] como uma ferramenta para permitir a implementação de um sistema de gerenciamento de rede distribuído [BAROTTO 98] entre as máquinas que compõem a rede, independente de plataforma e com mecanismo de instalação automática, visto que estará disponível através de *browsers Web*.

Apesar da linguagem de marcação HTML ser predominante nos documentos *Web*, suas conhecidas limitações vêm se evidenciando, levando alguns a considerar um momento de colapso para a HTML [HOLZNER 01]. Tal problemática, somada a grande inserção e aceitação da linguagem HTML como meio de publicação de documentos na *Web*, motivou o W3C a desenvolver a metalinguagem XML [HOLZNER 01] [GRAHAM 00].

O “esquema padronizado para a portabilidade dos dados”, sugerido por Case, pode ser efetivado com a tecnologia XML, que distingue entre interface, processos e dados. Além dessa distinção, XML oferece uma flexibilização no intercâmbio de dados; a possibilidade de personalizar linguagens de marcação, a estruturação, integração e autodescrição de dados. A característica fundamental de XML de dissociação entre estrutura de apresentação e conteúdo do documento, abre espaço para uma revolução na forma como as informações são manipuladas. Um mesmo documento base pode ser automaticamente convertido para diferentes formatos, podendo ser apresentado em um monitor de um computador pessoal, em uma pequena tela de um telefone celular ou até mesmo ser transformado em voz para utilização de deficientes visuais.

Sistemas de gerenciamento de redes que utilizam a *Web* como infra-estrutura de distribuição podem explorar as características de estruturação e flexibilização propiciadas por XML, produzindo documentos de informação de gerenciamento de redes consistentes e formalmente validados. Tais documentos constituem matéria-prima para as mais diferentes aplicações, podendo ser apresentados em inúmeros formatos nos *browsers Web* ou importados por outros sistemas de informação.

7.2 Trabalhos Futuros

Vislumbra-se como foco de estudo futuro a implementação e teste do modelo que emprega CORBA, a fim de avaliar suas vantagens potenciais, advindas da distribuição.

Referências Bibliográficas

1. [ADVENTNET 01] ADVENTNET. **AdventNet SNMP API Release 3.3**: Product Documentation. set. 2001.

2. [AHMED 98] AHMED, Suhail. **Corba Programming Unleashed**. Indianapolis: SAMS, 1998.

3. [BAROTTO 98] BAROTTO, André Mello. **Realização da Gerência Distribuída de Redes Utilizando SNMP, Java, WWW e CORBA**. Florianópolis, abr. 1998. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística – Universidade Federal de Santa Catarina.

4. [BAROTTO 00] BAROTTO, André Mello; SOUZA, Adriano de; WESTPHALL, Carlos Becker. **Distributed Network Management Using SNMP, Java, WWW and CORBA**. Journal of Network and Systems Management, v. 8, n.4, set., p. 251-265, 2000.

5. [BARILLAUD 97] BARILLAUD, Frank; DERI, Luca; FERIDUN, Metin. **Network Management Using Internet Technologies**. Fifth IFIP/IEEE Internet Symposium on Integrated Network Management, San Diego, CA, USA, p. 61-70, 1997.

6. [BERG 99] BERG, Clifford. **Advanced Java 2 Development for Enterprise Applications.** New Jersey: Prentice Hall, 1999.

7. [BOSAK 99] BOSAK, Jon; BRAY, Tim. **XML and the Second-Generation *Web***. Scientific American, New York, mai.,1999. Acessado em 11 jul. 2001. Online. Disponível na Internet em *http://www.sciam.com/1999/0599issue/0599bosak.html* .

8. [CASTRO 01] CASTRO, Elizabeth. **XML para a World Wide Web**. Rio de Janeiro: Campus, 2001.

9. [COMER 92] COMER, Douglas E.; STEVENS, David L. **Internetworking with TCP/IP**. Vol. I. ,New Jersey: Prentice Hall, 1992.

10. [COMER 99] COMER, Douglas E.; STEVENS, David L. **Interligação em Rede com TCP/IP**. Vol. II. Rio de Janeiro: Campus, 1999.

11. [DEROSE 99] DEROSE, Steven. **The SGML FAQ Book**: Understanding the Foundation of HTML and XML, Kluwer Academic Publishers, 1999.

12. [DEITEL 01] DEITEL, H. M.; DEITEL, P. J. **Java Como Programar**. Porto Alegre: Bookman, 2001. 3ª edição.

13. [DERI 99] DERI, Luca Haj. ***Web*-Acessible Network Managements Tools**. International Journal of Network Management, v. 9, 371–378, nov., 1999.

14. [ECKEL 01] ECKEL, Bruce. **Thinking in Java**. New Jersey: Prentice Hall, 2001. 2ª edição.

15. [FLANAGAN 97] FLANAGAN, David. **Java in a Nutshell**. New Jersey: O'Reilly, 1997. 2ª edição.

16. [FURGERI 01] FURGERI, Sérgio. **Ensino Didático da Linguagem XML**. São Paulo: Érica, 2001.

17. [GOSLING 96] GOSLING, James; JOY, Bill; STEELE, Guy. **The Java Language Specification**. Berkeley : Addison-Wesley, 1996.

18. [GRAHAM 00] GRAHAM, Ian S. **XHTML 1.0**: *Web* Development Sourcebook. New York : John Wiley & Sons, 2000.

19. [HAROLD 01] HAROLD, Elliotte Rusty; MEANS, W. Scott. **XML in a Nutshell**: A Desktop Quick Reference. New Jersew: O'Reilly, 2001.

20. [HARKEY 98] HARKEY, Dan; ORFALI, Robert. **Client/Server Programming with Java and CORBA, 2nd Edition**. New York : John Wiley & Sons, 1998.

21. [HAGGERTY 98] HAGGERTY, Paul; SEETHARAMAN, Krishnan. **The Benefits of CORBA-Based Network Management**. Communications of the ACM, v. 41, 10, 73–79, out., 1998.

22. [HOLZNER 01] HOLZNER, Steven. **Desvendando XML**. Rio de Janeiro: Campus, 2001.

23. [IBM 01] IBM.**XML**:Education. Acessado em 11 jul. 2001. Online. Disponível na Internet em *http://www-106.ibm.com/developerworks/education/xmlintro/xmlintro.html* .

24. [INTRANET 02] INTRANET PARANÁ. **Intranet Paraná**. Acessado em 06 mar. 2002. Online. Disponível em *http://www.intranetparana.br/*.

25. [ISO 86] ISO. ISO 8879:1986 **Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML)**. 1986

26. [ISO 95] ISO. ISO 10746-1:1995 **Reference Model of Open Distributed Processing** – Part 1. 1995.

27. [JANDER 96] JANDER, M. **Welcome to the Revolution**. Data Communications International, nov., 1996.

28. [JU 02] JU, Hong-Taek; et al. **An Embedded Web Server Architecture for XML-Based Network Management**. Proceedings of IEEE/IFIP Network Operations And Management Symposium (NOMS). Florença, Italia. Maio, 2002.

29. [JU 01] JU, Hong-Taek; CHOI, Mi-Jung; HONG, James W. **EWS-Based Management Application Interface and Integration Mechanisms for Web-Based Element Management**. Journal of Network and Systems Management, v. 9, n. 1, 31-50, 2001.

30. [KOPKA 99] KOPKA, Helmut; DALY, Patrick. **A Guide to Latex :** Document Preparation for Beginners and Advanced Users. Berkeley : Addison-Wesley, 1999. 3ª edição.

31. [LEWIS 01] LEWIS, David; MOURITZSEN, Jens D. **The Role of XML in TMN Evolution**. Proceedings of IEEE/IFIP International Symposium on Integrated Network, 689-702, 2001.

32. [MARCHAL 00] MARCHAL, Benoît. **XML By Example**. Indianápolis: QUE, 2000.

33. [MARTIN-FLATIN 00] MARTIN-FLATIN, J. P. **Web-Based Management of IP Networks and Systems**. Out. 2000. Ph.D. Thesis, EPFL, Lausanne, Switzerland.

34. [MCGRATH 98] MCGRATH, Sean. **Rendering XML Documents Using XSL**. Dr. Dobb's Journal, n. 287, 82-85, jul., 1998.

35. [MCGRATH 99] MCGRATH, Sean. **XML**: Aplicações Práticas. Rio de Janeiro: Campus, 1999.

36. [McLAUGHLIN 01] McLAUGHLIN, Brett. **Java and XML**. New Jersew: O'Reilly, 2001. 1ª edição.

37. [MULLER 97] MULLER, Nathan J. ***Web*-Acessible Network Managements Tools**. International Journal of Network Management, v. 7, 288–297, set., 1997.

38. [NAKHIMOVSKY 00] NAKHIMOVSKY, Alexander; MYERS, Tom. **Professional Java XML Programming with Servlets and JSP**. Birmingham: Wrox Press, 2000. 1ª edição.

39. [NMS 02] NMS. **Network Management Server (NMS)**. Acessado em 13 jul. 2002. Online. Disponível na Internet em http://netman.cit.buffalo.edu/index.html.

40. [OMG 01] OMG. **Object Management Group**. Acessado em 08 jul. 2001. Online. Disponível na Internet em *http://www.omg.org* .

41. [ORFALI 99] ORFALI, Robert; HARKEY, Dan; EDWARDS, Jeri. **Client/Server Survival Guide, 3rd Edition**. New York : John Wiley & Sons, 1999.

42. [RAY 01] RAY, Erik T. **Learning XML** : (Guide to) Creating Self-Describing Data. New Jersew: O'Reilly, 2001.

43. [ROSE 90] ROSE, M.; MCCLOGHRIE, K. **Structure and Identification of Management Information for TCP/IP based internets**. RFC 1155. Network Working Group. 1990.

44. [SEBESTA 00] SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. Porto Alegre: Bookman, 2000. 4ª edição.

45. [STALLINGS 98] STALLINGS, Willian; 1998. **SNMP and SNMPv2**: The Infrastructure for Network Management. IEEE Communications Magazine. n. 168, 37-43, mar., 1998.

46. [STALLINGS 00] STALLINGS, William. ***SNMP, SNMPv2, SNMPv3, and RMON 1 and 2***. 3 Ed. Reading, MA: Addison Wesley, 2000.

47. [THOMPSON 98] THOMPSON, P. ***Web*-Based Enterprise Management Architecture.** IEEE Communications Magazine, v. 36, n. 3, mar., 1998.

48. [UC 01] UNICODE CONSORTIUM (UC). **What Is Unicode?**. Acessado em 15 jul. 2001. Online. Disponível na Internet em *http://www.unicode.org/unicode/standard/WhatIsUnicode.html*.

49. [VERONEZ 99] VERONEZ, C. A.; EFRAIN, C.; BAROTTO, A. M.; NASSAR, S. M.; WESTPHALL, C. B. **Gerência de Redes Utilizando Métodos Estatísticos Bayesianos**. Anais do Simpósio Brasileiro de Redes de Computadores. Salvador (BA), Brasil. Maio 1999.

50. [VLIST 01] VLIST, Eric Vander. **XML Schema**. New Jersew: O'Reilly, 2001.

51. [XMLINFO 01] **XMLINFO**: The XML Information Site. Acessado em 11 jul. 2001. Online. Disponível na Internet em *http://xmlinfo.com* .

52. [XML.ORG 01] **XML.ORG**: The XML Industry Portal. Acessado em 23 mai. 2001. Online. Disponível na Internet em *http://www.xml.org/xmlorg_resources/index.shtml* .

53. [W3C 01a] WORLD WIDE WEB CONSORTIUM (W3C). **Extensible Markup Language (XML)**. Acessado em 5 mar. 2001. Online. Disponível na Internet em *http://www.w3.org/XML/* .

54. [W3C 01b] WORLD WIDE WEB CONSORTIUM (W3C). **Extensible Markup Language (XML) 1.0 (Second Edition)**. Acessado em 2 abr. 2001. Online. Disponível na Internet em *http://www.w3.org/TR/REC-xml*.

Anexo A – Código-Fonte Java

```
/**
 * Title:        SNMPMan
 * Description:
 * Copyright:    Copyright (c) 2002
 * Company:      Unioeste
 * @author Fabio Alexandre Spanhol
 * @version 1.0
 */
// pacotes Java
import javax.swing.*;
import org.netbeans.lib.awtextra.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;
// pacotes da API SNMPAdventNet
import com.adventnet.snmp.snmp2.*;
import com.adventnet.snmp.snmp2.usm.*;
import com.adventnet.snmp.beans.*;

class GetData  {


   final String Host;
   final String OID;


   javax.swing.JTextArea JTA_Status;

   private int ONE_SECOND = 1000;    /* valor de um segundo, expresso em milésimos de
segundo        apenas para melhorar a legibilidade do código-fonte */
   private javax.swing.Timer timerPooling;

   private boolean firstCatch = true;
   private boolean errorFound = false;

   long previousTime;

   Double  previousValue;



   SnmpTarget target = new SnmpTarget(); /* instancia o bean SnmpTarget
        permite uma sessão SNMP leve para
    comunicação SNMP síncrona */
   GetData ( String Host, String OID) {
      this.Host = Host;
      this.OID  = OID;
      /* Intervalos regulares de 5 minutos */
      timerPooling = new javax.swing.Timer(ONE_SECOND * 300, new ActionListener() {
            public void actionPerformed(ActionEvent evt) {

               String response = "";

               Date now = new Date();

               long actualTime  = 0,
                    bytes       = 0,
                    elapsedTime = 0;


               response = snmpGet ();

               if (!errorFound) {

                      if (firstCatch) {

                         previousTime  = now.getTime ();
                      previousValue = new Double (response);
```

```
                    firstCatch    = false;

                  } else {

                        actualTime = now.getTime();
                        Double actualValue = new Double (response);

                        elapsedTime = (actualTime - previousTime)/(1000);//em segundos


                        bytes       = actualValue.longValue() -
previousValue.longValue();

                        System.out.println ("Bytes:" + bytes);


                    long rateBits = (bytes ) * 8 / elapsedTime;


                    JTA_Status.append ("\n->OID [" + getOID() + "]" + rateBits + "
bps");

                   }
                } else {
                     JTA_Status.append("\n" + response);
                }
            }
         });
   }

   String getOID () {

     return (this.OID);
   }

   String getHost () {

          return (this.Host);

   }



        void start () {
               Date date = new Date();

          if (timerPooling.isRunning())
             timerPooling.restart();
          else
             timerPooling.start();

          JTA_Status.append("\n->Pooling de coleta iniciado ...OID [" + getOID() + "] " +
date.toLocaleString() + "\n");

       }


       void stop  () {
         Date  date = new Date();


         timerPooling.stop();
         JTA_Status.append("\n->Pooling de coleta interrompido..." +
date.toLocaleString() + "\n");

       }


       String snmpGet  () {
               target.setTargetHost(this.Host);         //seta o host no qual o agente
SNMP está rodando
               target.setObjectID (this.OID);           // seta o OID
               String result = target.snmpGet(); // executa uma requisição SNMP GET NEXT
               //Verifica se houve erro
```

```
            if (result == null) {
                //Requisição falhou ou houve time out
                errorFound =  true;
                return ( target.getErrorString());


        } else {
                errorFound = false;
                return (result);
        }
  }
}
public class SNMPMan extends javax.swing.JFrame  {

   GetData in,   //objeto para consultar informações do tráfego de entrada
           out;  //objeto para consultar informações do tráfego de saida

   //declaração de variáveis para tornar o código-fonte mais legível
   public final String Default_Router = "200.201.8.1",
                       ifOutOctets    = ".1.3.6.1.2.1.2.2.1.16.4",
                              ifOutDiscards  = ".1.3.6.1.2.1.2.2.1.19.4",
                              ifOutErrors    = ".1.3.6.1.2.1.2.2.1.20.4",
                              ifInOctets     = ".1.3.6.1.2.1.2.2.1.10.4",
                              ifInDiscards   = ".1.3.6.1.2.1.2.2.1.13.4",
                              ifInErrors     = ".1.3.6.1.2.1.2.2.1.14.4";

   /* determina como será a aparência da interface da aplicação
          para o padrão MS-Windows a String deve ser
"com.sun.java.swing.plaf.windows.WindowsLookAndFeel" */
   public static String windowsClassName =
"com.sun.java.swing.plaf.motif.MotifLookAndFeel";
   /* Objetos da interface gráfica com o usuário*/
   private javax.swing.JMenuBar menuBar;
   private javax.swing.JMenu fileMenu;
   private javax.swing.JMenu helpMenu;
   private javax.swing.JMenuItem getMenuItem;
   private javax.swing.JMenuItem stopMenuItem;
   private javax.swing.JMenuItem exitMenuItem;
   private javax.swing.JMenuItem aboutMenuItem;
   private javax.swing.JTextArea JTA_Status;
   private static javax.swing.JTextField JTF_Host;
   private javax.swing.JLabel JL_Host;
   private javax.swing.JLabel JL_Status;
   private javax.swing.JDesktopPane JDP_Panel;
   public SNMPMan () {
     initComponents();
     initMyComponents();
   }
   private void initComponents() {

      JDP_Panel = new javax.swing.JDesktopPane();
      JTA_Status= new javax.swing.JTextArea();

      JTF_Host  = new javax.swing.JTextField();
      JL_Host   = new javax.swing.JLabel();
      JL_Status = new javax.swing.JLabel();

      menuBar = new javax.swing.JMenuBar();

      fileMenu = new javax.swing.JMenu();
      helpMenu = new javax.swing.JMenu();

      getMenuItem   = new javax.swing.JMenuItem();
      stopMenuItem  = new javax.swing.JMenuItem();
      exitMenuItem  = new javax.swing.JMenuItem();
      aboutMenuItem = new javax.swing.JMenuItem();
      //------- menu Arquivo   ----------
      fileMenu.setText("Arquivo");
      fileMenu.setMnemonic('A');

      getMenuItem.setText ("Iniciar Coleta");
      getMenuItem.setMnemonic ('I');
      getMenuItem.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            getMenuItemActionPerformed(evt);
         }
      });
      stopMenuItem.setText ("Finalizar Coleta");
      stopMenuItem.setMnemonic ('F');
      stopMenuItem.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
            stopMenuItemActionPerformed(evt);
         }
      });

      fileMenu.addSeparator();

      exitMenuItem.setText("Sair");
      exitMenuItem.setMnemonic('S');
      exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            exitMenuItemActionPerformed(evt);
         }
      });
      //-------- menu Ajuda ------------
      helpMenu.setText("Ajuda");
      helpMenu.setMnemonic('u');

      aboutMenuItem.setText("Sobre");
      aboutMenuItem.setMnemonic('o');
      aboutMenuItem.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            aboutMenuItemActionPerformed(evt);
         }
      });
      fileMenu.add (getMenuItem);
      fileMenu.add (stopMenuItem);
      fileMenu.add(exitMenuItem);

      helpMenu.add(aboutMenuItem);

      menuBar.add(fileMenu);
      menuBar.add (helpMenu);
      getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
      setTitle("SNMPManager vers\u00e3o 1.0");
      addWindowListener(new java.awt.event.WindowAdapter() {
         public void windowClosing(java.awt.event.WindowEvent evt) {
            exitForm(evt);
         }
      });
     //----------------------------------------
      JL_Host.setText("Roteador Monitorado:");
      JL_Host.setForeground(new java.awt.Color(30, 19, 143));
      JL_Host.setFont(new java.awt.Font("Verdana", 3, 12));
      getContentPane().add(JL_Host, new rg.netbeans.lib.awtextra.AbsoluteConstraints(10,
5, -1, -1));
      JTF_Host.setForeground(new java.awt.Color(255, 51, 51));
      JTF_Host.setFont(new java.awt.Font("Dialog", 1, 12));

      JTF_Host.setText(Default_Router);
      JTF_Host.setEditable(false);

      getContentPane().add(JTF_Host, new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 25, 175, -1));


      JDP_Panel.setBorder(new javax.swing.border.EtchedBorder());


     /* JDP_Panel.setBackground(new java.awt.Color(174, 178, 195));
      JDP_Panel.setAutoscrolls(true);
      getContentPane().add(JDP_Panel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(190, 30, 640, 450)); */


      JL_Status.setText("Situa\u00e7\u00e3o");
      JL_Status.setFont(new java.awt.Font("Verdana", 2, 12));
```

```
      getContentPane().add(JL_Status, new
org.netbeans.lib.awtextra.AbsoluteConstraints(190, 5, 150, -1));
      JTA_Status.setEditable(false);
      JTA_Status.setForeground(new java.awt.Color(255, 51, 51));
      JTA_Status.setBackground(new java.awt.Color(255, 255, 255));

      JTA_Status.setFont(new java.awt.Font("Dialog", 3, 12));
      JTA_Status.setBorder(new javax.swing.border.EtchedBorder());
      JTA_Status.setAutoscrolls(true);
      getContentPane().add(JTA_Status, new
org.netbeans.lib.awtextra.AbsoluteConstraints(192, 25, 430, 380));
      JScrollPane JSP_Scroll5 = new JScrollPane(JTA_Status);
      JSP_Scroll5.setAutoscrolls(true);
      this.getContentPane().add(JSP_Scroll5, new
org.netbeans.lib.awtextra.AbsoluteConstraints(192, 25, 430, 380));
      JTA_Status.setText("->Coleta em Estado Ocioso...");
     //---------------------------------------
      setJMenuBar(menuBar);
      pack();
      java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
      //setSize(new java.awt.Dimension(1024, 768));
      setSize(new java.awt.Dimension(640, 480));
      setLocation((screenSize.width-640)/2,(screenSize.height-480)/2);
      try {
         UIManager.setLookAndFeel(windowsClassName);
         SwingUtilities.updateComponentTreeUI(this);
      }
      catch (Exception exc) {
         System.err.println("Error loading Look and Feel: " + exc);
      }

  }
  void initMyComponents () {

       out = new GetData (Default_Router, ifOutOctets);

       in = new GetData (Default_Router, ifInOctets);

       in.JTA_Status = this.JTA_Status;
       out.JTA_Status = this.JTA_Status;


  }


  private void getMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
       in.start();
       out.start();

  }
  private void stopMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
       in.stop();
       out.stop();
  }
  private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
      System.exit(0);
  }
  private void exitForm(java.awt.event.WindowEvent evt) {
      System.exit(0);
  }
  private void aboutMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
  }
  public static void main (String args[]) {
       new SNMPMan().show();
  }
}
```

Anexo B – Documento XML Gerado pela Aplicação

```
<?xml version="1.0" encoding="UTF-8"?>
<LOG xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\Usr\Fabio\Mestrado CPGCC\SNMPManager\vld_doc.xsd">
<MNG>
       <HOST ip="200.201.8.1">Roteador</HOST>
       <OID>.1.3.6.1.2.1.2.2.1.10.4</OID>
       <SNMPVERSION>3</SNMPVERSION>
       <DATE>2002-08-27</DATE>
       <TIME>15:18</TIME>
       <RESPONSE unit="bps">881654</RESPONSE>
</MNG>
<MNG>
       <HOST ip="200.201.8.1">Roteador</HOST>
       <OID>.1.3.6.1.2.1.2.2.1.10.4</OID>
       <SNMPVERSION>3</SNMPVERSION>
       <DATE>2002-08-27</DATE>
       <TIME>15:23</TIME>
       <RESPONSE unit="bps">905536</RESPONSE>
</MNG>
<MNG>
       <HOST ip="200.201.8.1">Roteador</HOST>
       <OID>.1.3.6.1.2.1.2.2.1.10.4</OID>
       <SNMPVERSION>3</SNMPVERSION>
       <DATE>2002-08-27</DATE>
       <TIME>15:28</TIME>
       <RESPONSE unit="bps">883968</RESPONSE>
</MNG>
</LOG>
```

Anexo C – XMLSchema com a estrutura do documento XML Coletado

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="qualified">

	<!-- Define o tipo utilizado no atributo ip  -->
	<xsd:simpleType name="ipaddressType">
		<xsd:restriction base="xsd:string">
			<xsd:pattern value="\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"/>
		</xsd:restriction>
	</xsd:simpleType>

	<!-- Define o tipo complexo utilizado no elemento HOST -->
	<xsd:complexType name= "hostType" mixed="true">
	   <xsd:attribute name="ip" type="ipaddressType" use="required"
value="200.201.8.1" />
	</xsd:complexType>

	<!-- Define o tipo  utilizado no elemento SNMPVERSION -->
	<xsd:simpleType name="snmpversionType">
		<xsd:restriction base="xsd:positiveInteger">
			<xsd:minInclusive value="1"/>
			<xsd:maxInclusive value="3"/>
		</xsd:restriction>
	</xsd:simpleType>

	<!-- Define o tipo complexo utilizado no elemento RESPONSE -->
	<xsd:complexType name= "responseType" mixed="true">
	   <xsd:attribute name="unit" type="xsd:string" use="required" value="bps" />

	</xsd:complexType>

	<!-- Define o tipo complexo utilizado no elemento MNG -->
	<xsd:complexType name="MngType">
		<xsd:sequence>
			<xsd:element name="HOST" type="hostType" />
			<xsd:element name="OID" type="xsd:string"/>
			<xsd:element name="SNMPVERSION" type="snmpversionType"/>
			<xsd:element name="DATE" type="xsd:date"/>
			<xsd:element name="TIME" type="xsd:time"/>
			<xsd:element name="RESPONSE" type="responseType"/>
		</xsd:sequence>
	</xsd:complexType>

      <!-- Define o tipo complexo utilizado no elemento raiz LOG -->
      <xsd:complexType name="logType">
           <xsd:sequence>
                <xsd:element name="MNG" type="MngType" minOccurs="1"
maxOccurs="unbounded"/>
           </xsd:sequence>
      </xsd:complexType>

	<!-- Elemento raiz do documento -->
	<xsd:element name="LOG" type="logType" />
</xsd:schema>
```