FUNDAÇÃO DE ENSINO EURÍPIDES SOARES DA ROCHA
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ IVO FERNANDES DE OLIVEIRA

DESEMPENHO ARQUITETURAL DE COMPONENTES GRÁFICOS WIMP USANDO O PADRÃO SVG EM APLICAÇÕES WEB

MARÍLIA
2007

JOSÉ IVO FERNANDES DE OLIVEIRA

DESEMPENHO ARQUITETURAL DE COMPONENTES GRÁFICOS WIMP USANDO O PADRÃO SVG EM APLICAÇÕES WEB

Dissertação apresentada ao Programa de Mestrado Stricto Sensu em Ciência da Computação do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do título de Mestre em Ciência da Computação (Área de Concentração: Arquitetura de Computadores).

Orientador: Prof. Dr. Edward David Moreno Ordonez.

MARÍLIA
2007

José Ivo Fernandes de Oliveira

Desempenho arquitetural de componentes gráficos WIMP usando o padrão SVG em aplicações WEB

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM / FEESR, para obtenção do Título de Mestre em Ciência da Computção.

A Comissão Julgadora:

Prof. Dr. Edward David Moreno Ordonez

Prof. Dra. Kalinka R. L. J. Castelo Branco

Prof. Dr. Sérgio Takeo Kofuji

Marilia, 30 de agosto de 2007.

Formatado: Justificado
Formatado: Esquerda
Formatado: Justificado
Formatado: Fonte: Itálico
Formatado: Esquerda
Formatado: Fonte: Itálico
Formatado: Português
Formatado: Esquerda

AGRADECIMENTOS

Agradeço a Deus pela proteção divina e por me ajudar diante de vários obstáculos além de me dar força para seguir sempre em frente.

Aos meus familiares, filhos, netos e, em especial, à minha mãe e o meu pai (em memória), que sempre me apoiaram nos momentos de desânimo, de alegria e que me deram estímulos para continuar meus estudos.

A todos os meus amigos, em especial aos Professores: Prof. Dr. José Carlos, à Profª. Msc. Renata Paschoal e ao Prof. Msc. Fábio Dacêncio Pereira, que pela amizade e pela troca de experiências, sem dúvida alguma, me ajudaram a evoluir como ser humano e como profissional na área da docência.

Aos professores pelo conhecimento transmitido. Ao Prof Dr. Marcos Mucheroni pelos conselhos, e um agradecimento em especial ao meu orientador, Prof. Dr. Edward David Moreno Ordonez, uma pessoa maravilhosa que sempre transmitiu, não só a mim, mas a todos os seus orientados, apoio, carinho, ajuda e compreensão nos momentos difíceis.

OLIVEIRA, José Ivo Fernandes de. **DESEMPENHO ARQUITETURAL DE COMPONENTES GRÁFICOS WIMP USANDO O PADRÃO SVG EM APLICAÇÕES WEB.** 135 f. José Ivo Fernandes de Oliveira; Marilia, SP, 2007. Dissertação de Mestrado em Ciência da Computação – Centro Universitário Eurípides de Marília. Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

Formatado: Cor da fonte: Preto

Excluído: 27

Formatado: Cor da fonte: Vermelho

RESUMO

Este trabalho tem o propósito de demonstrar o impacto arquitetural na utilização de componentes WIMP presentes na interface gráfica de usuário (GUI) usando a tecnologia SVG, observando o consumo do processador, memória e latência da rede das imagens vetoriais e raster em aplicações *Web*. Com o uso destes componentes gráficos de interação semânticos (usando SVG) baseado no padrão W3C em navegadores, espera-se uma redução no custo computacional dos elementos essenciais da *Web*, principalmente em servidores e na otimização da largura da banda da internet habilitando apenas a transmissão de dados incrementais entre o cliente e o servidor. Também se espera ter visualizações com uma maior definição (imagem vetorial), flexibilidade, portabilidade para o processamento gráfico no ambiente da internet. No decorrer do trabalho foi possível analisar a carga do processador (tanto do cliente como do servidor), o tempo de aquisição e o uso da memória em três diferentes navegadores (Internet Explorer, Firefox e Opera), apontando o melhor navegador e as vantagens do uso do formato SVG.

Palavras-chave: Custo Computacional**,** Imagens Vetoriais, SVG, WIMP.

OLIVEIRA, José Ivo Fernandes de. **DESEMPENHO ARQUITETURAL DE COMPONENTES GRÁFICOS WIMP USANDO O PADRÃO SVG EM APLICAÇÕES WEB.** 135 f. José Ivo Fernandes de Oliveira; Marilia, SP, 2007. Dissertação de Mestrado em Ciência da Computação – Centro Universitário Eurípides de Marília. Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

Formatado: Cor da fonte: Preto

Excluído: 27

Formatado: Cor da fonte: Vermelho

ABSTRACT

This work is intended for demonstrating the architectural impact in the use of WIMP components present in user's graphic interface (GUI) using the technology SVG, and observing the consumption of the processor, memory and latency of the net of vectorial and raster images in *Web* applications. With the use of these semantic interaction graph components (using SVG) based on the W3C pattern in navigators, a reduction is expected in the computational cost of the essential elements of the *Web*, especially in servers and in the optimization of the bandwidth of the internet, enabling only the transmission of incremental data between client and server. It is also expected visualizations with a larger definition (vectorial image), flexibility and portability for the graphic processing into the Internet environment. When developing this work it was possible to analyze the load of the processor (as much of the client as of the server), the time of acquisition and the use of the memory in three different navigators (Internet Explorer, Firefox and Opera), pointing the best navigator and the advantages of the use of SVG.

Keywords: Computational Cost, Vectorial Images, SVG, WIMP.

LISTA DE FIGURAS

LISTA DE TABELAS

Excluído: 74
Excluído: 74
Excluído: 83
Excluído: 89
Excluído: 95
Excluído: 100
Excluído: *Tabela 1 -* ... [1]
Excluído: 40
Excluído: ¶ ... [2]
Excluído: 98
Inserido: 61
Excluído: ¶ ... [3]
Inserido: 98
Excluído: ¶ ... [4]
Inserido: 110
Excluído: 116
Inserido: 111
Excluído: 117
Excluído: ¶ ... [5]
Excluído: 118
Excluído: ¶ ... [6]
Inserido: Erro! Indicado ... [7]
Inserido: 58
Excluído: 60
Excluído: ¶ ... [8]
Excluído: 122
Inserido: 58
Excluído: 60
Excluído: ¶ ... [9]
Excluído: 63
Inserido: 93
Excluído: 76
Excluído: ¶ ... [10]
Excluído: 104
Inserido: 72
Excluído: ¶ ... [11]
Inserido: 72
Excluído: 76
Excluído: ¶ ... [12]
Excluído: ¶ ... [13]
Inserido: 81
Excluído: 85
Inserido: 112
Excluído: ¶ ... [14]
Inserido: 87
Excluído: 92
Excluído: ¶ ... [15]
Inserido: 113
Excluído: 119
Excluído: ¶ ... [16]
Excluído: ¶ ... [17]
Inserido: 114
Excluído: 120
Inserido: 115
Inserido: 116
... [18]
... [19]

LISTA DE ABREVIATURAS E SIGLAS

ActiveX	Interface usada para entrada e saída de dados em aplicação Microsoft.
AI	Adobe Illustrator.
AIX	Sistema Operacional da IBM.
API	Application Programming Interfaces .
BATIK	Ferramenta Baseada em SVG rever.
Beans	Componente Java.
BMP	Extensão do arquivo BMP.
BNF	Notação Backus Naur Forms.
Browser	Programa cliente que é usado para visualizar Aplicativo na *Web*.
CLI	Command Line Interface.
Cliente	Aplicação utilizado para conectar e obter dados de um servidor.
CDR	Extensão do arquivo CorelDraw.
CGM	Computer Graphics Metafile.
COM	Component Objetct Model.
CPU	Unidade Central de Processamento.
CSS	**C**ascading **S**tyle **S**heet.
DCOM	Distributed Component Object Model.
DOM	Document Object Model.
DTD	Document Type Definition.
DXF	Extensão de imagem do AutoCAD.
ECMA	European Computer Manufactures Association.
GIF	Graphics Interchange Format.
GUI	Graphics User Interface.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
IDL	Interface Definition Language.
Icons	Pequenas imagens com significados e significantes.
IP	Internet Protocol.
ISA	Instruction Set Architecture.
ISO	International Standard Organization.
IIOP	Internet Inter-Orb Protocol.
JAVA	Linguagem de programação orientada.

Excluído: para uma

Formatado: Italiano (Itália)

Formatado: Fonte: Negrito

Formatado: Francês (França)

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

JPEG	Joint Photographers Expert Group.
JRMI	Java Remote Method Invocation.
W3C	Word Wide Web Consortium.
WEB	Rede Mundial de Computadores.
WIMP	Windows, Icon, Menu and Pointer.
LZW	Lempel-Ziv-Welch.
MVC	Model-View-Controller.
M-MVC	Message-Model-View-Controller.
OMG	Object Management Group.
ORB	Object Request Broker.
PDA	Personal Digital Assistant.
PNG	Portable Network Graphics.
RPC	Remote Procedure Call.
SWF	Extensão do arquivo Shockwave Flash.
SMIL	Synchronized Multimedia Integration Language.
SOCKET	Aplicação cliente servidor.
SVG	Scalable Vector Graphics.
Thread	Parte do resultado de um processo.
Toolkits	Conjunto de Ferramentas.
TUI	Tangible User Interface.
UI	User Interface.
UCSD	User Centered System Development.
XSL	Extensible Stylesheet Language.
ZOS	Sistema Operacional Main Frame IBM.

Excluído: s

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

SUMÁRIO

Excluído: 121213151517181923252931333738414244454647485154576465666767696972737373798181868888929397101108110118121126126INTRODUÇÃO . 12¶
1.1 MOTIVAÇÃO . 12¶
1.2 Objetivos da dissertação . 14¶
1.3 Organização da dissertação . 14¶
2. Tecnologias para Sistemas Distribuídos . 16¶
2.1 Apresentação das Tecnologias de Componentes . 17¶
2.2 Tecnologia da arquitetura CORBA . 18¶
2.3 Modelo de Componente Distribuído (DCOM) . 22¶
2.4. Computação Distribuída na Linguagem Java . 24¶
2.4.1 Componente Java Beans . 28¶
2.4.2 Socket na linguagem Java . 30¶
2.4.2.1 Modos de operações do componente Socket . 31¶
2.4.3 Applet . 32¶
2.4.3.1 Desenvolvimento de Applets . 33¶
2.5 Computação Distribuída usando a Linguagem C . 36¶
2.6 Considerações finais do capítulo . 38¶
3. Fundamentação das Interfaces Gráficas GUI . 42¶
3.1 Novas Metáforas: Interfaces e Tecnologia . 43¶
3.2 Simbiose homem/máquina e interfaces de Usuário . 45¶
3.2.1 Linguagem natural e por comando . 46¶
3.2.2 Bases da Teoria da Cognição e a Linguagem . 47¶
3.2.3 Interface de linha de Comando (CLI) . 48¶
3.3 Interface Gráfica de Usuário – (GUI) . 49¶
3.4 Componentes de Interação WIMP. . 52¶
3.5 Interface Tangível de Usuário (TUI) . 55¶
3.6 Interfaces gráficas com SVG para Web . 58¶
3.7 Considerações finais do Capítulo . 66¶
4. Análise Arquitetural de Componentes Gráficos WIMP em SVG . 67¶
4.1 Definições das métricas . 68¶
4.1.1 Métrica adotada - Tempos de respostas . 69¶
4.1.2 Captura da Carga do Processador . 70¶
4.1.3 Utilização da memória . 71¶
4.2 Metodologia e infra-estrutura utilizada . 72¶
4.3 Ocupação do Processador . 75¶
4.4 Resultados Obtidos . 76¶
5. Conclusão . 102¶
REFERÊNCIAS BIBLIOGRÁFICAS . 10
... [20]

Inserido: 121213151517181923252931333738414244454647485154576465666767696972737373798181868888929397101108110118121126126

INTRODUÇÃO

Os padrões atuais das Interfaces Gráficas de Usuário (GUI) possuem origens históricas comuns: Xerox, Apple Macintosh, IBM SAA, MIT X-Windows System, todas elas são fundamentadas em imagens do tipo *raster* (APPLE, 1992), (PARC, 2006). Essas interfaces gráficas consomem uma significativa parcela do poder computacional, acarretando um maior custo em relação a outras interfaces não gráficas (CHANG, 1990).

Aliado ao custo computacional das interfaces gráficas, três elementos essenciais presentes na arquitetura da rede podem afetar o desempenho de aplicativos da *Web*: servidores menos potentes, meio de transmissão deficiente e máquinas clientes com baixo poder computacional (MENASCÉ, 2002), (TANENBAUM, 2003).

Assim, a possível redução desse custo incentiva a incorporação do padrão de Interfaces Gráficas de Usuário baseada na tecnologia SVG *(Scalable Vectors Graphics)* (QIU, 2005) que poderá contribuir com melhorias para os sistemas distribuídos, tais como: desempenho, amigabilidade, escalabilidade dos sistemas e principalmente no impacto da utilização da rede.

Pierre Dragicevic (DRAGICEVIC, 2004) destaca em seu trabalho que a evolução do *software* tornou-se cada vez mais complexa e especializada e que as interfaces de usuário não acompanharam essa evolução. O autor também afirma que várias pesquisas apontam que é possível obter uma interação mais natural e produtiva aplicada ao contexto de interação usuário-computador.

1.1 O formato SVG

A definição do padrão SVG[1] foi realizada pelo consórcio W3C sem fins lucrativos e que define os padrões abertos. Várias organizações inclusive acadêmicas são responsáveis pela criação das linguagens de marcação HTML e XML dentre outros vocabulários. São diversas organizações, envolvidas na definição do SVG como, por exemplo: o MIT *Massachusetts Institute of Technology,* a Apple, Sun MicroSystems, o INRIA *Institut National de Recherche en Informatique et en Automatique,* Adobe, IBM, Kodak e a Keio University.

[1] *O acrônimo **SVG** significa Scalable Vector Graphics (Gráficos Vetoriais Escaláveis) gráficos expressos em XML.*

SVG é uma linguagem criada e recomendada pelo consórcio W3C, que descreve gráficos bidimensionais (retas e curvas) expressos em relações matemáticas em XML. A primeira especificação foi a (SVG 1.0) no ano de 2001 (HISTORY, 2006).

A especificação do padrão **SVG** divide-se em:

- ***Full*** - completa que engloba as especificações *Mobile* e *Print*;
- ***Mobile*** - voltada para equipamentos móveis (Celular, PDAs e Palms);
- ***Print*** - destinada à impressão.

Esse formato gráfico SVG é flexível em sua aplicação, podendo ser usado em conjunto com outras tecnologias como, por exemplo: JavaScript, *scripting* no lado servidor, XML e recursos de estilo como CSS - ***C****ascading* ***S****tyle* ***S****heet*[2], XSL - *eXtensible Stylesheet Language*[3], criando assim várias possibilidades de utilização em aplicações de engenharia, científicas, negócios e interface de usuário GUI.

O conjunto destes conceitos define as capacidades do SVG e justifica a sua criação como a tecnologia de gráficos para a *World Wide Web*.

A linguagem SVG permite a existência combinada de três tipos de objetos gráficos:

• formas gráficas vetoriais (polígonos, curvas, linhas);

• imagens *raster* e texto.

Todos estes objetos podem ser agrupados, transformados. Todos os gráficos em SVG podem ser interativos ou serem animados através de elementos SVG ou de *scripts* externos. Uma das vantagens desse formato gráfico é o fato de não sofrerem efeito de *aliasing*[4] podendo ser redesenhados sem sofrer distorção.

Formatado: Fonte: Itálico

1.2 MOTIVAÇÃO

Esta dissertação realizou um estudo sobre os componentes gráficos de interações semânticos do tipo WIMP, baseados em SVG – Gráfico Vetorial Escalar, enfatizando a carga na aquisição e distribuição de aplicações que habilitem a interoperabilidade e escalabilidade, que refletem a perspectiva sobre a Internet e os diversos serviços em desenvolvimento para uma nova *Web*.

---

[2] *Folha de estilo em cascata é um mecanismo simples para adicionar estilos aos documentos Web.*
[3] *Define a formatação e apresentação de documentos XML.*
[4] *Distorção da imagem.*

A origem dessa dissertação surgiu dos vários trabalhos pesquisados, dentre os quais se destacam:

- Xiaohong Qiu (QIU, 2005) demonstrou a utilização do componente SVG no modelo proposto MMVC - Modelo de Visão e Controle baseado em Mensagem;
- Pierre Dragicevic (DRAGICEVIC, 2001) mostrou a Interação do usuário na seleção e a configuração de dispositivos com o sistema Icon; o mesmo autor, em (DRAGICEVIC, 2004) onde ele apresentou um modelo de multi-interação com diversos dispositivos de entrada altamente configuráveis TUI - Interface Tangível de Usuário;
- Brygg Ullmer (ULLMER, 2001) descreveu a relação entre a representação física e a informação digital, destacando os vários tipos de interação no modelo TUI - Interface Tangível de Usuário;
- Rodrigo García (GARCIA, 2006), em seu artigo, descreveu a aplicabilidade dos componentes gráficos em SVG no sistema SCADA *(Supervisory Control and Data Acquisition)*. Nesse sistema é disponibilizado uma interface gráfica com wimp necessária para monitorar e controlar processos de produção em uma planta industrial;
- Felipe Marinho (MARINHO et al., 2006) mostrou a utlização de componentes gráficos de interação em SVG no projeto GENESys. Este projeto utiliza a *Web* para monitorar e controlar sistemas energéticos geograficamente distribuídos.

A substituição da interface em linha de comando pela interface gráfica facilitou em muito a amigabilidade dos sistemas por parte dos usuários. Porém, essa evolução exigiu um maior poder computacional de processamento voltado para esse novo ambiente.

Compreendendo a natureza desse trabalho em termos do desempenho das interfaces gráficas no ambiente da *Web*, avalia-se o seu impacto em diversas arquiteturas heterogêneas e, finalmente, busca-se uma solução que venha reduzir o custo computacional nos diversos ambientes colaborativos.

1.3 Objetivos da dissertação

Embora a tecnologia do componente SVG seja ainda nova (1999-2000) (HISTORY, 2006), as indústrias de tecnologias desenvolveram várias ferramentas (Apache Batik, SVG Toolkit, Inkscape, Sketsa, EvolGrafiX, etc.) para a construção e aplicação desse componente visando a usabilidade e interoperabilidade dos sistemas. Diante deste contexto, ressalta-se a necessidade de estudar esta tecnologia que enfrenta o desafio de integrar diversas aplicações (abordando os aspectos práticos e o impacto na arquitetura), além de examiná-la e propor as adaptações se necessárias que contribuam para o seu amadurecimento.

Os objetivos principais dessa dissertação incluem:

Construir componentes gráficos de interação WIMP com tecnologia SVG utilizando os recursos da linguagem XML - *Extensible Markup Language*, combinado com a linguagem Javascript, visando comparar o seu impacto na arquitetura cliente/servidor no processamento distribuído e os possíveis ganhos na sua utilização;

Além disso, avaliou-se o desempenho desses componentes de interação gráficos semânticos fundamentados na tecnologia SVG, analisando o impacto no meio de transmissão, no servidor e no cliente em função do tamanho do pacote transmitido. O trabalho verifica também métricas como a utilização do processador, necessidade de memória, tanto do cliente como servidor.

Finalmente, comparam-se os diferentes formatos de imagens baseadas em *raster* (GIF, PNG, JPG e BMP) com o formato SVG e suas principais características nas transações entre um cliente e um servidor, monitorando os tempos de resposta ponta a ponta – latência inicial e interpretando os resultados obtidos do custo do processamento dentre outros.

1.4 Organização da dissertação

Além da introdução, esta dissertação possui quatro capítulos:

O capítulo 2 conceitua e apresenta as Tecnologias para Sistemas Distribuídos e a descrição dos principais componentes distribuídos. Já o capítulo 3 apresenta a fundamentação

e o histórico das interfaces gráficas. O capítulo 4 relata a análise arquitetural de componentes gráficos WIMP em SVG.

O capítulo 5 apresenta as conclusões desse estudo em aspectos tais como: ocupação da CPU e da memória pelo formato SVG, e ainda sugere alguns assuntos para trabalhos futuros.

2. Tecnologias para Sistemas Distribuídos

Com a evolução das redes de computadores surgiram as aplicações distribuídas. No início, o processamento era realizado somente no servidor e as aplicações eram inicializadas por um cliente que estava interligado a ele por meio da rede. Mais tarde, com o surgimento da programação orientada a objetos surgiram novos *middlewares* como CORBA, DCOM e os componentes derivados das linguagens C e JAVA, nos quais o processamento passou a ser distribuído entre vários servidores (TANENBAUM, 2002).

Atualmente, com o avanço da Internet e dos protocolos de comunicação, além de novos padrões das linguagens de marcação como o XML, surgiram os serviços da *Web* com a proposta de integrar sistemas heterogêneos por meio de novas tecnologias.

A tecnologia para sistemas distribuídos refere-se à padronização dos elementos que são utilizados em programas ou aplicações que fazem chamadas de ativação remotamente a outros aplicativos, que residem em outros computadores e que estão em diferentes domínios interligados em uma rede (COULOURIS, 1998).

A finalidade dos sistemas distribuídos é facilitar o alcance dos objetivos por um grupo de usuários utilizando recursos do sistema (*hardware* e *software*). Para tanto, é fundamental que os diferentes tipos de equipamentos e sistemas operacionais existentes no ambiente se comuniquem de uma forma eficaz, rápida e segura (COULOURIS, 1998), (TANENBAUM, 2002).

A computação de objetos distribuídos é uma estrutura para a computação cliente/servidor. É essencial destacar que essa estrutura disponibiliza e dá suporte à troca e ao reúso de objetos distribuídos, facilitando aos desenvolvedores construir sistemas a partir da montagem de componentes de diversos fabricantes (COULOURIS, 1998).

No modelo de objetos distribuídos, a arquitetura define um módulo que fica entre os clientes e o servidor. Nesse padrão, objetos clientes requisitam serviços às implementações dos objetos que estão no lado servidor através de um determinado componente, o qual é responsável por mapear todos os mecanismos requeridos para encontrar o objeto solicitado, preparar a implementação do objeto para receber a requisição do cliente, e finalmente executá-la. O cliente visualiza a requisição independentemente de onde esteja o componente e da linguagem de programação que tenha sido concebido (TANENBAUM, 2002).

Os aplicativos na arquitetura cliente/servidor são dispostos em duas camadas: aplicativo cliente e servidor (COULOURIS, 1998), (TANENBAUM, 2002). Essas camadas são partes integrantes do aplicativo e elas podem estar em diferentes domínios ou não. Todos os protocolos de componentes distribuídos são construídos com características semelhantes para serem utilizados em uma arquitetura básica da rede.

Os serviços disponibilizados por um ambiente distribuído se caracterizam por serem provedores de serviços para que os usuários possam utilizá-los. Mas para que isto ocorra, um formato padrão deve ser compreensível para qualquer um que necessite utilizá-los em um registro central que esteja disponível (COULOURIS, 1998).

Sobre essa arquitetura são executadas operações fundamentais para utilização dos diferentes serviços: localização, regras de negócios, interações e protocolos. Cada uma dessas operações precisa ser padronizada para que o serviço seja utilizado por qualquer aplicação, independente do tipo de linguagem e da plataforma onde esteja sendo executada. A próxima seção mostrará as diversas tecnologias para arquitetura distribuída.

2.1 Apresentação das Tecnologias de Componentes

Segundo Andrew Tanenbaum (TANENBAUM, 2003), as tecnologias de componentes para integralizar o desenvolvimento de *software* têm se revelado uma abordagem muito útil e com inúmeros benefícios. Dentre dos vários, pode-se destacar: o encapsulamento (modularidade), o polimorfismo (flexibilidade) e a herança (reusabilidade), características essas essenciais para a qualidade e a produtividade do desenvolvimento de *software*.

Considerando a heterogeneidade de plataformas computacionais e de sistemas operacionais presentes nas grandes redes cooperativas e a Internet, a visão de uma aplicação deve considerar a possibilidade de utilização de componentes (ou objetos) concebidos nas diferentes linguagens e capazes de serem executados em diferentes ambientes (COULOURIS, 1998), (TANEMBAUM, 2002).

Objetivando proporcionar um ambiente de desenvolvimento capaz de manipular a heterogeneidade e a distribuição das aplicações atuais, foi necessário que o OMG *(Object Management Group)* definisse uma arquitetura. A arquitetura definida, CORBA *(Common Object Request Broker Architecture)*, (OMG, 2006) é uma arquitetura de suporte ao desenvolvimento voltado para aplicações distribuídas heterogêneas orientadas a objetos.

Desde o início dos anos 90, quando foram publicados os primeiros documentos relativos à arquitetura CORBA, um número significativo de trabalhos foi iniciado no sentido de aprimorar essa arquitetura e de apresentar implementações compatíveis com a sua especificação.

Mais recentemente, a Microsoft apresentou a especificação de um ambiente com a mesma orientação do CORBA, a arquitetura DCOM, que tem sido vista como uma proposta concorrente àquela proposta pelo OMG.

Um modelo de implementação nesse paradigma apresenta a linguagem JAVA, que é uma arquitetura OPEN SOURCE, estimula as recentes pesquisas.

Na presente seção é mostrada as principais características e conceitos relativos às arquiteturas CORBA, DCOM e JAVA.

A seção 2.2 apresenta a arquitetura CORBA e os seus principais elementos. Já seção 2.3 descreve o componente DCOM, na seção 2.4 apresenta a linguagem JAVA e seus recursos para implementação de componentes distribuídos. Na seção 2.5 mostra-se os recursos de implementação na linguagem C.

2.2 Tecnologia da Arquitetura CORBA

CORBA é uma arquitetura definida pelo OMG e tem sua base no modelo cliente/servidor e no paradigma de orientação a objetos. Segundo este modelo, a execução de uma aplicação distribuída segue um esquema de interação entre clientes e servidores de modo que um cliente encaminha solicitações de serviços a objetos CORBA (IDL. 2006).

As solicitações de serviços, assim como as respostas a estes, são gerenciadas por um ambiente de comunicação denominado ORB *(Object Request Broker).* Para cada objeto de uma aplicação existe a especificação dos serviços que ele pode oferecer. O cliente, de posse desta informação, é capaz de encaminhar sua solicitação ao objeto que poderá atendê-la (TANENBAUM, 2003).

Os objetos CORBA podem assumir o papel de clientes de outros objetos CORBA quando executarem um de seus serviços que tenha sido anteriormente solicitado por um cliente, devendo fazer uso de um ou diversos serviços providos por outros objetos. O cliente

pode obter informações dos serviços do objeto nas interfaces dos mesmos. Estas interfaces tanto podem fornecer estas informações ao cliente quanto à infra-estrutura de comunicação (TANENBAUM, 2003).

O formato das mensagens recebidas e enviadas pelos componentes não fica visível, de modo que é necessário assegurar a transparência na comunicação entre o cliente e o objeto servidor. Estes objetos recebem um identificador único para diferenciá-los um do outro, independente do local onde se encontrem.

Nesse sentido, a arquitetura OMA *(Object Management Architecture)* foi criada pelo consórcio OMG (SUN, 2006) com o objetivo de integrar as aplicações baseadas em objetos distribuídos. Sua estrutura é baseada essencialmente em dois elementos: o modelo de objeto que define o objeto que será distribuído pelo sistema heterogêneo e o modelo de referência que define as características da integração destes objetos.

O componente CORBA possui toda sua estrutura baseada na arquitetura OMA. Esta estrutura define a comunicação entre os objetos distribuídos por meio de quatro elementos básicos:

- objetos de aplicação: são os objetos criados pelo usuário e possuem características definidas de acordo com os seus objetivos. Também são tidos como os usuários finais de toda a infra-estrutura CORBA;
- facilidades comuns: definem as regras de integração para que os objetos possam colaborar efetivamente;
- serviços comuns: são serviços que possibilitam a implementação e utilização desses objetos. Eles definem uma estrutura de baixo nível, ampliando as funcionalidades do ORB e são utilizados para criar um objeto, nomeá-lo e introduzi-lo no ambiente.
- barramento de objetos: é a base dessa arquitetura. Ela é denominada de ORB *(Object Request Broker)* que é o *middleware* que estabelece os relacionamentos do servidor com o cliente, sendo através dele que os objetos de *software* se ligam para interoperarem.

O corretor ORB (*Broker*) intercepta o atendimento e é responsável pela localização de um objeto que possa executar a solicitação e passar os parâmetros, além de invocar seu método e retornar seus resultados sem que o cliente tenha conhecimento de onde está

localizado este objeto, qual a sua linguagem de programação implementada ou o seu sistema operacional.

Desta forma, a estrutura de comunicações do ORB permite que os objetos interajam, independentes das técnicas utilizadas para a sua implementação e dos aspectos específicos da plataforma (TANENBAUM, 2003).

Esse corretor permite que os objetos transmitam ao usuário solicitações em um ambiente distribuído e heterogêneo de forma transparente.

Vários serviços opcionais foram definidos pelo OMG que podem ser disponibilizados, tais como: localização de objetos pelo nome, manutenção de objetos persistentes, suporte ao processamento de transações, comunicação, segurança e outras facilidades atuais presentes nos ambientes da computação distribuída. Ressalta-se que a linguagem Java suporta a maioria desses serviços adicionais (BORUSCH et al., 2005), (CORBA, 2006).

As aplicações clientes e objetos que se comunicam no CORBA podem ser escritas em diferentes linguagens de programação. Para que se efetive a comunicação é necessário que exista uma interface chamada IDL *(Interface Definition Language)* (IDL, 2006).

A primeira etapa para desenvolver uma aplicação CORBA é definir a sua interface IDL. Na IDL é realizado o mapeamento de linguagem de programação para IDL (para Java, C, C++) e são incluídas as definições dos tipos de dados específicos de cada linguagem e interfaces para se acessar os objetos CORBA.

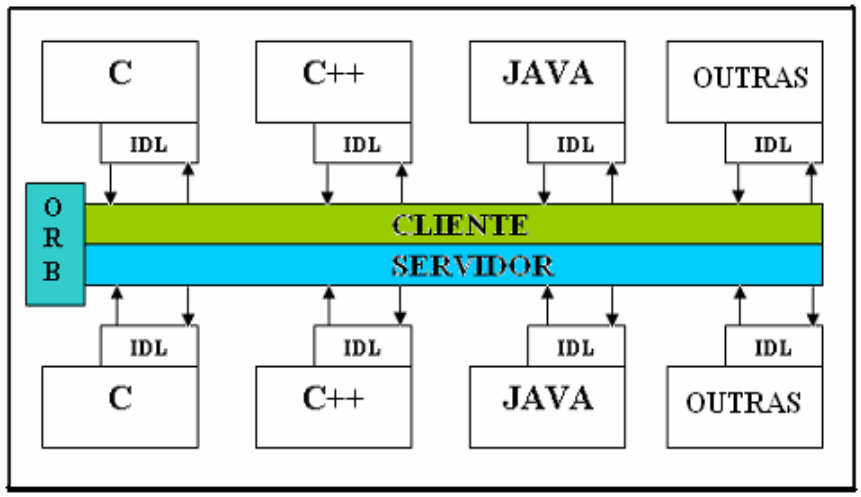

Figura 1 - Interface para diferentes Linguagens de Aplicação (CORBA, 2006).

Esses objetos podem ser distribuídos em diferentes plataformas, tais como: Unix, Win32, Linux, FreeBSD etc. A figura 1 ilustra o esquema da utilização da IDL, com a qual o usuário tem contato direto, e esta com o componente que está no lado servidor.

A transparência dos fatores citados acima é responsável não apenas pela garantia como também pela ativação de um componente para o qual uma solicitação é encaminhada. Esta ativação é realizada de forma transparente, não importando se o componente está situado em nível local ou remoto (IDL, 2006).

Essa ativação ocorre como se fosse uma requisição, como outra qualquer, realizada pelo cliente, ao qual retorna a referência objeto. Isto faz com que a estrutura do ORB permaneça bem simples, que é uma das características interessantes do CORBA, se preocupando somente com a comunicação e a infra-estrutura de ativação para as aplicações distribuídas, deixando para os demais objetos do OMA o máximo de funcionalidade possível (TANENBAUM, 2003).

Os elementos responsáveis pela invocação dos métodos de chamadas remotas são os *stubs e skeletons.* As funções realizadas pelos *stubs* e os *skeletons* são correspondentes. O *stub* fica situado na extremidade do cliente e o *skeleton* na extremidade do servidor de objetos. Eles implementam a arquitetura cliente/servidor no CORBA.

Tanto o *stub* quanto o *skeleton* são criados com base na compilação da interface IDL do cliente e do objeto, respectivamente. Quando um cliente emite uma solicitação, ele só precisa dizer ao *stub* para qual componente ela é destinada, independente do componente estar situado a nível local ou remoto, conforme ilustra a figura 2.

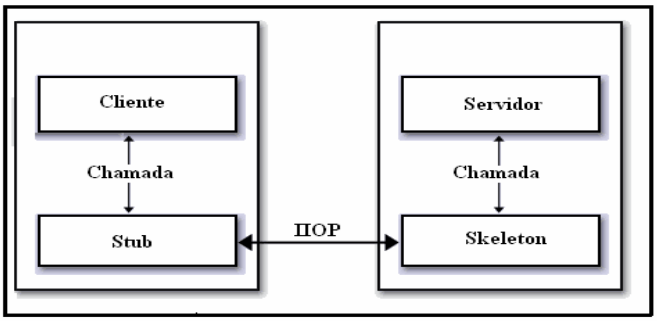

Figura 2 - Funções do Stub e Skeleton (CORBA, 2006).

O acesso remoto é feito através do protocolo IIOP (*Internet Inter-ORB*). Esse protocolo faz com que os programas distribuídos escritos em diferentes linguagens se comuniquem através do ambiente de uma rede corporativa ou da internet (CORBA, 2006).

2.3 Modelo de Componente Distribuído (DCOM)

O surgimento do componente DCE *(Distributed Computer Environment)* da Microsoft foi o marco inicial da tendência de computação distribuída nas empresas. Na sua evolução surgiu o componente DCOM *(Distributed Component Object Model)*: um protocolo que possibilita os componentes de *software* a se comunicarem diretamente sobre uma rede de modo confiável, seguro e eficiente, utilizando múltiplos protocolos de transporte de rede, incluindo protocolos de Internet, por exemplo, o protocolo HTTP *(Hyper Text Transfer Protocols).*

O DCOM é baseado na especificação DCE-RPC da *Open Software Foundation*. Ele trabalha em conjunto com applets Java e componentes ActiveX por meio do uso do Component Object Model.

Os objetos e interfaces COM são especificados pelo uso da IDL (Linguagem de Definição da Interface) da Microsoft, uma extensão do padrão DCE IDL (COM) (TANENBAUM, 2003).

Esse componente é uma extensão do *Component Object Model* (COM). O modelo define como os componentes e os seus clientes interagem. Esta interação é definida de maneira que o cliente e o componente podem ser interligados sem a necessidade de qualquer sistema intermediário. O cliente invoca métodos do componente sem qualquer sobrecarga.

Nos sistemas operacionais os processos estão protegidos uns dos outros. Um cliente que necessite trocar informações com outro componente distante não o pode solicitar diretamente, ou seja, é necessário que o sistema operacional proporcione qualquer forma de comunicação entre os processos.

O componente COM fornece essa comunicação de uma maneira completamente transparente: intercepta as chamadas de um cliente e reencaminha-as para um componente em outro processo. O COM/DCOM fornece a ligação entre o cliente e o componente. Quando o cliente e o componente residem em diferentes máquinas, o componente DCOM simplesmente substitui a comunicação entre processos local pelo protoloco de rede, mostrando a sua funcionalidade independente de sua localização.

Essa interação é definida de maneira que o cliente e o componente possam se conectar sem a necessidade de qualquer outro recurso intermediário (PLÁSIL, 1999).

A tecnologia DCOM permite a comunicação em diversos protocolos de rede: TCP/IP, UDP/IP, IPX/SPX, Appletalk e HTTP. Os componentes DCOM normalmente são escritos em JAVA, porém podem ser escritos em C (TANENBAUM, 2003).

A independência de localização desse componente nessa tecnologia simplifica a tarefa de distribuir componentes para melhorar o desempenho no lado cliente, supondo, por exemplo, que certos componentes devam estar em uma determinada máquina e em uma determinada localização.

Caso a aplicação possuir muitos componentes pequenos é possível reduzir o tráfego na rede colocando-os nos mesmos segmentos, na mesma máquina ou até no mesmo processo. Se a aplicação possuir um grande número de componentes grandes e o tráfego na rede não for problema, então, é preferível colocá-los em máquinas mais rápidas, onde quer que elas residam (TANENBAUM, 2002).

Com a característica proporcionada pelo DCOM de independência e de localização a aplicação pode combinar componentes relacionados em uma única máquina ou até no mesmo processo. Mesmo que exista um número grande de pequenos componentes que em conjunto implementam um módulo lógico maior, eles podem continuar interagindo de forma eficiente.

Os componentes podem ser executados na máquina que fizer mais sentido: interface do usuário e validação perto do cliente ou regras intensivas de acesso à base de dados de um servidor próximo da base de dados.

Com destaque a neutralidade de linguagem das aplicações, os desenvolvedores podem escolher as linguagens com as quais estejam mais familiarizados. A independência de linguagem também permite que se criem protótipos rapidamente: os componentes podem ser desenvolvidos inicialmente numa linguagem de alto nível, como o Microsoft Visual Basic, Delphi e mais tarde ser reimplementado em outra linguagem, como o C++ ou Java, por exemplo.

2.4. Computação Distribuída na Linguagem Java

A Empresa Sun Microsystems em meados de 1990 desenvolveu uma linguagem de programação derivada da linguagem C++ denominada Java. Os desenvolvedores da Sun

corrigiram uma série de problemas que apresentavam na linguagem C++, por exemplo, a linguagem Java utiliza algoritmo coletor de lixo *(garbage collector)* para deslocar regiões de memória que não estejam mais sendo usadas. Por ser orientada a objeto, também permite o desenvolvimento de aplicativos de forma mais rápida, o que significa uma economia de tempo e de mão de obra.

Essa linguagem também permite ao programador desenvolver um aplicativo que possa ser executado em qualquer plataforma: Linux, Unix, FreeBSD, Windows, AIX, Macintosh, etc, desde que o computador tenha um interpretador que faça a interface entre o usuário e a aplicação (HOPSON, 1997), (CORBA, 2006).

O ambiente de programação da linguagem Java disponibiliza uma máquina que é implementada como a combinação de uma máquina real e de um *software* de virtualização. Ela pode ter recursos diferentes da máquina real, ou em quantidade ou em tipo. Por exemplo, uma máquina virtual pode ter mais ou menos processadores que a máquina real, os quais podem executar uma instrução diferente do que a fixada na máquina real.

É importante salientar que o desempenho equivalente normalmente não é requerido como parte da virtualização. Freqüentemente, uma máquina virtual provê menos desempenho que uma máquina real equivalente que execute o mesmo *software*, por exemplo, a execução de um aplicativo desenvolvido para a máquina real (JAVA, 2005). A figura 3 ilustra uma abstração da Máquina Virtual Java comparada com a máquina real.

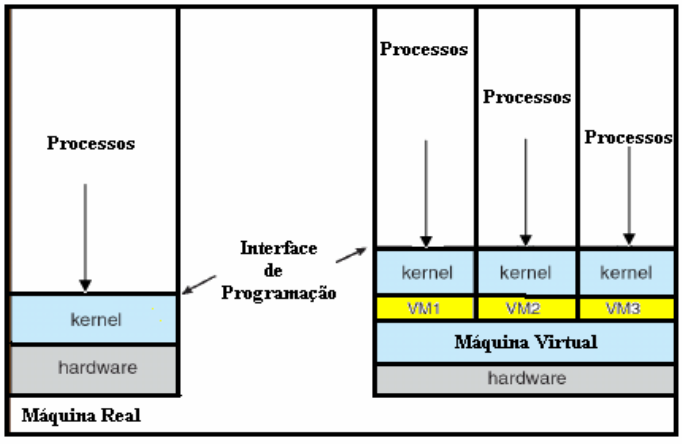

Figura 3 - Abstração da Máquina Virtual Java (TANENBAUM, 2003).

Existe uma grande variedade de máquinas virtuais que fornecem também uma grande variedade de benefícios. Múltiplas máquinas virtuais podem ser implementadas e reproduzidas em uma única plataforma de *hardware* para prover os usuários que se agrupam com o próprio ambiente do sistema operacional.

Um servidor de multi processadores pode ser dividido em servidores menores virtuais desde que garantam a habilidade para nivelar o uso de recursos de *hardware* pelo sistema.

As máquinas virtuais também podem empregar técnicas de emulação para apoiar compatibilidade de *software*. Os níveis de abstrações em um sistema de computador correspondem a camadas de implementação em *hardware* e *software*. Existe uma arquitetura para cada uma destas camadas, embora o termo "arquitetura" não seja sempre usado (SMITH et al,. 2005).

Na linguagem Java, as APIs (*CosNaming*, CORBA, *PortableServer, rmi.Remote*, etc) disponibilizam os recursos necessários ao desenvolvimento e são compatíveis com os padrões CORBA, RMI, JRMI de objetos distribuídos, com o propósito de atender a solicitação de um serviço cliente/servidor com interopabilidade e portabilidade.

Os objetos distribuídos são ferramentas poderosas que só ficaram amplamente difundidos e disponíveis nos últimos anos devido à larga expansão dos serviços disponíveis da *Web*. Esses objetos podem interagir diretamente com um objeto que se mantenha em um *host* remoto. O poder de distribuir objetos está no fato de que uma aplicação atender a diversos usuários simultaneamente em um ambiente de rede (TANENBAUM, 2002).

Um simples exemplo de aplicação em Java IDL, obtidos da documentação da Sun (CORBA, 2005), é uma implementação do conhecido e tradicional *"hello world",* composto por três arquivos (vide apêndice A) que exemplificam o modelo de serviço cliente/servidor utilizando a arquitetura CORBA. Vide figura 4 (IDL, 2006).

- Um servidor que cria um objeto e o publica com o nome do serviço, usando o lado servidor nessa aplicação;
- Uma aplicação cliente que invoca as referências do objeto ORB.

A interface IDL define como os métodos de diferentes aplicativos desenvolvidos em diferentes linguagens podem acessar os serviços de um componente. Neste exemplo, o código Hello.idl possui essa função. Para executar essa aplicação, é necessário seguir a seqüência abaixo:

1) compilar a IDL com o compilador IDLJ com o seguinte parâmetro: idlj –fall Hello.idl. Após a compilação várias classes serão criadas: HelloPOA, _HelloStub,

Hello, HelloHelper, HelloHolder e HelloOperations. Essas classes irão disponibilizar as funcionalidades básicas do padrão CORBA. A função básica de cada uma é:

- **HelloPOA.java**: Essa classe abstrata é o Skeleton do fluxo de dados do servidor e provê a funcionalidade básica do CORBA pelo servidor, que se conecta com a interface de HelloOperations. A classe de servidor HelloImpl é estendida da classe HelloPOA;
- **_HelloStub.java:** Essa classe é o *stub* do cliente e provê a funcionalidade do CORBA para o cliente;
- **Hello.java**: Essa interface contém a versão da linguagem Java na interface da IDL. A interface do arquivo Hello.Java provê a funcionalidade do padrão CORBA. Também é uma interface estendida da classe HelloOperations;
- **HelloHelper.java**: Essa classe possui a função de ler e escrever os dados digitados;
- **HelloHolder.java**: Essa classe final assegura se um tipo de dado está ou não na semântica da linguagem Java na IDL;
- **HelloOperations.java**: Essa interface contém o *sayHello* dos métodos das operações definidas na interface de IDL que neste arquivo é compartilhado pelos *stub* (cliente) e *skeleton* (servidor).

2) compilar todos os arquivos Java, incluindo o *stub* e o *skeleton*, no diretório criado pelo IDLJ. Nesta etapa, a linguagem Java incluirá o diretório helloapp. E utilizar o seguinte comando: javac *.java HelloApp/*.java;
3) inicializar o servidor com a seguinte instrução: orbd -ORBInitialPort 1050 (porta 1050);
4) inicializar a classe Hello server com a seguinte instrução: java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost;
5) executar a aplicação cliente com o seguinte comando: java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost.

Apesar de ser um simples projeto, o aplicativo *helloword* exemplifica todas as tarefas para desenvolver qualquer aplicação que venha a utilizar o padrão CORBA, uma inovação estática que é usada pelo *stub* do cliente para a solicitação do serviço no *skeleton* do servidor.

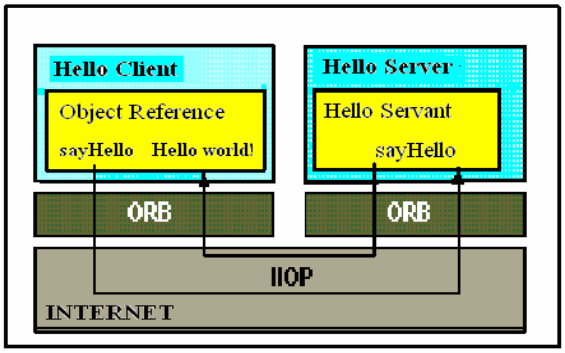

Figura 4 - Arquitetura da aplicação cliente/servidor em java (IDL, 2006).

Esse exemplo requer um serviço denominado de *naming* que permite ao CORBA criar um vínculo de um nome a uma referência do objeto. A ligação do nome pode ser armazenada no serviço *naming*, e um cliente pode usá-lo para obter a referência do objeto desejado. A figura 5 ilustra esse vínculo.

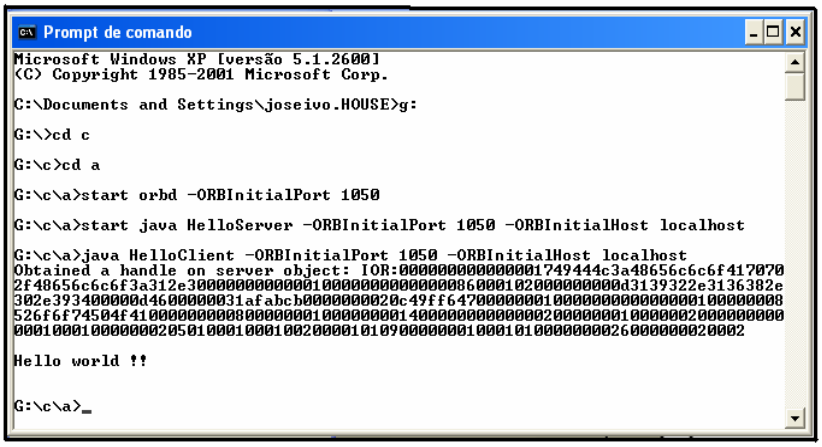

Figura 5 - Mensagem enviada pelo servidor ao cliente (IDL, 2006).

Além das características e vantagens técnicas mencionadas, a linguagem Java é poderosa para a modelagem de objetos a partir dos recursos fornecidos pelo paradigma da orientação a objetos que acompanham a evolução do *software* de uma maneira mais transparente, facilitando a sua manutenção (IDL, 2006).

2.4.1 Componente Java Beans

A empresa Sun Microsystems define o componente JavaBean como um componente de *software* reutilizável e de fácil desenvolvimento, até mesmo para aplicações complexas. É escrito na linguagem Java utilizando uma IDE Java Beans (Eclipse, NetBeans). A programação é feita da mesma forma que se escreve qualquer outra aplicação na linguagem Java, facilitando o acréscimo de outros componentes existentes, por exemplo, *applets* ou aplicação Java, transformando-os em componentes JavaBeans (BEANS, 2006).

Esse componente em sistemas distribuídos pode compartilhar informações, mas é necessário definir os mecanismos que permitam estabelecer a sua integração. Uma vez integrados, os sistemas podem além de compartilhar informações, compartilhar também suas funcionalidades.

As tarefas semelhantes ou até mesmo idênticas não precisam ser reimplementadas em um ambiente de sistemas integrados. O ideal é que estas tarefas sejam desenvolvidas de maneira que se tornem componentes compartilhados. Para que esta integração ocorra, é necessária a existência de uma arquitetura de *software* robusta, flexível e que permita a escalabilidade e a evolução de todos os sistemas envolvidos neste contexto sem a necessidade de refazer cada sistema sempre que surgir uma nova funcionalidade.

Esses componentes Beans foram criados para facilitar a implementação de aplicações que servem a vários usuários, em um ambiente distribuído e com operações que suportem transações da lógica específica do negócio. Eles disponibilizam acesso concorrentemente aos dados, pois foram projetados com as seguintes características: gerenciam acessos concorrentes; possuem recursos *multi-thread* automático; disponibilizam a reusabilidade; portabilidade e separação bem definida da interface com a implementação. Com relação à portabilidade, eles podem ser executados em diferentes plataformas de *hardware* e sistemas operacionais heterogêneos.

Essas vantagens diminuem os riscos de uma aplicação tornar-se obsoleta quando houver a atualização da plataforma visto que possuem as seguintes características: segurança na troca de informações; controle de falhas; gerenciamento de retorno “religação” quando acontecer uma falha no servidor; escalabilidade em aplicações que necessitem o aumento da demanda por incremento de novos clientes de serviços; acesso permitido a aplicativos desenvolvidos em diferentes linguagens, por exemplo: clientes Java podem acessar esses componentes por meio das interfaces padrões. Clientes não Java também podem acessar os Beans usando o CORBA com total transparência (BEANS, 2006).

Para caracterizar um componente Java Beans são necessários pelo menos três elementos fundamentais: a interface cliente; a interface remota e a classe Bean.

A interface cliente define os métodos que permitem ao cliente criar, remover ou encontrar um Bean; a interface remota define quais métodos de negócio um cliente pode solicitar e o Bean implementa os métodos de negócio devolvidos ao cliente pela interface remota (BEANS, 2006).

A ativação de um processo é iniciada a partir da solicitação do cliente para o componente EJB, que mantém a referência remota ao cliente, e para respondê-la, cria-se uma conexão com o cliente via *stub*. Quando o cliente invoca um método de negócios no objeto EJB é necessário obter uma informação do contêiner que chama o método *ejbActivate()* e, em seguida, os campos devem ser sincronizados com um banco de dados. A partir de então, o método *ejbLoad()* é executado para notificar ao Bean que os campos estão corretos e, por fim, as requisições remotas do cliente podem ser atendidas. O contêiner é responsável pelo controle de todo o fluxo da operação descrita.

Todos esses elementos envolvidos no processo têm suporte das seguintes APIs: javax.ejb, javax.servlet, javax.servlet.jsp, javax.naming, java.sql, java.transaction e javax.jms. (BEANS, 2006).

A tecnologia Java Beans ou simplesmente Beans assume uma grande importância no cenário da integralização. A sua arquitetura segue o padrão dos componentes distribuídos e a linguagem Java oferece suporte para o desenvolvimento desse componente.

2.4.2 Socket na Linguagem Java

Um *socket* é um ponto final de uma comunicação entre aplicativos que são executados em uma rede (DEITEL. 2005). Ele está ligado a um número de porta de maneira que a camada do protocolo TCP pode identificar a aplicação cliente para que os dados sejam enviados.

Normalmente, um *socket* é executado em uma máquina servidora. Uma vez ativado, o servidor fica pronto, esperando para ouvir as requisições do cliente (o pedido de conexão, por exemplo). No lado cliente o aplicativo que utiliza o *socket* possui o endereço IP e o número da porta que o aplicativo servidor está "escutando" (HOPSON, 1997), (SOCKET, 2006).

Para fazer um pedido de conexão o cliente também precisa se identificar para o que o serviço seja nomeado pelo sistema.

O funcionamento de um *socket* faz parte do protocolo TCP/IP (*Transfer Control Protocol*) usado na *Web*. Um *socket* é essencialmente uma conexão de dados transparente entre dois computadores em uma rede.

O *socket* é identificado pelo endereço da rede dos computadores e uma porta em cada computador. Os computadores em rede direcionam os fluxos de dados recebidos da rede para os aplicativos receptores específicos, associando cada aplicativo a um número diferente, a porta do aplicativo (HOPSON, 1997), (SOCKET, 2006).

Da mesma forma, quando o tráfego de saída é criado, o aplicativo de origem recebe um número de porta para a troca de dados. Caso contrário, o cliente poderia não responder à entrada.

O *socket,* a exemplo das outras tecnologias de objetos distribuídos, também está associado a um número de porta, tornando possível que a camada do protocolo TCP localize a aplicação que deverá receber os dados;

Cada computador que está conectado a uma rede possui um endereço único. As portas representam conexões privativas dentro desse endereço. Cada porta de um computador compartilha o mesmo endereço, mas os dados são distribuídos dentro de cada computador pelo número da porta (DUMAS, 1995), (SOCKET, 2006).

Destaque que a linguagem Java disponibiliza facilidades para o desenvolvimento de aplicações cliente/servidor. O pacote: import java.net possui as classes s*ockets* que devem ser definidas tanto no lado do servidor quanto no lado do cliente, objetivando efetuar as transmissões nas conexões para a troca de mensagens. No lado do servidor definem-se as propriedades da conexão e o gerenciamento das requisições. Por exemplo: processar informações de um banco de dados consultando ítens, atualizando cadastro.

Existem dois modos de operações com os *socket*: orientado e não orientado a conexões. O s*ocket* fundamentado em conexões possui a função de estabelecer uma conexão e encerrá-la. Já o não orientado a conexão opera sem a garantia de recebimento.

A operação orientada a conexão utiliza o protoclo TCP (*Transport Control Protocol*). Esse modo de operação necessita conectar-se ao destino antes de enviar os dados. Uma vez estabelecida a comunicação, o *socket* é acessado pelo uso de uma interface de fluxos de dados: leitura/escrita. Os dados transmitidos pelo *socket* são recebidos pelo cliente, exatamente na mesma ordem em que foram transmitidos. Apesar da operação baseada em

conexões ser menos eficiente do que a operação sem conexão ela é garantida (HOPSON, 1997), (COULOURIS, 1998), (SOCKET, 2006).

A operação não orientada a conexão utiliza o protocolo UDP (*User Datagrama Protocol*). Um datagrama é uma unidade que contém todas as informações necessárias para fazer sua entrega, como em um envelope (destinatário, endereço, CEP, remetente, e etc.), além de conter os dados a serem transmitidos no seu conteúdo. Essa maneira de operação não precisa se conectar a um *socket* de destino; ela simplesmente envia o datagrama. Apesar da operação sem conexão ser rápida e eficiente, ela não possui a garantia da entrega (HOPSON, 1997), (SOCKET, 2006).

A escolha do modo a ser utilizado vai depender da segurança exigida pelo aplicativo. É evidente que se a opção for pela não orientação à conexão com o servidor, deve-se pagar um custo maior de processamento na verificação dos dados (COULOURIS, 1998).

2.4.3 Applet

Um *applet* é um tipo específico de aplicação que está vinculado a um navegador *Web* (*Opera, Netscape Navigator, Internet Explorer, Mozilla Firefox, etc*) que interpreta o código Java, ou seja, o código binário que está sendo transportado pela rede. Para que seja possível, o documento HTML deverá conter uma referência a uma classe Java que se encontra no lado servidor e que será executada na máquina cliente como, por exemplo, nas seguintes tags:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Exemplo de Applet</title>
  </head>
  <body>
      <applet  code=ApptoAppl.class  archive=ApptoAppl.java
width=400 height=500>
      </applet>
  </body>
</html>
```

Outra característica do *applet* é que o mesmo não possui um método main() como as outras aplicações java (APPLETS, 2006).

Um *applet* implementa um conjunto de métodos que lidam com situações tais como inicialização, visualização do desenho da tela, eventos do mouse, etc. Os navegadores habilitados para Java se beneficiam do fato da linguagem ser dinâmica, colocando *applet* vinculando às páginas, carregando-o automaticamente quando essas páginas forem requisitadas, passando a fazer parte do navegador. As principais características do *applet* são:

- necessita de de um navegador para montar a sua interface gráfica;
- possui restrições de acesso a unidades de disco - só acessa com a permissão explícita do usuário;
- possui restrição de acesso à rede - podem acessar apenas o site do qual originaram;
- segurança;
- utiliza os métodos init( ), start( ), stop( ) e destroy( ) para definir seu ciclo de vida.

O método init ( ) é chamado imediatamente após a criação do *applet.* **Ele** é invocado pelo navegador na primeira vez que o *applet* é carregado. O método *start*( ) é chamado pelo navegador toda vez que o *applet* é concluído na tela. **O método** *stop*( ) é invocado pelo navegador toda vez que o *applet* deixa de ser visível. E, finalmente, o método *destroy*( ) é requisitado quando todos os recursos computacionais alocados pelo *applet* precisam ser liberados (APPLETS, 2006).

Na linguagem Java existe um conjunto de bibliotecas conhecido como Java API (Interface de Programação de Aplicações). Nessas bibliotecas está disponível um conjunto de classes organizadas em pacotes. Cada pacote traz as classes com a funcionalidade básica e necessária para um determinado tipo de aplicativo. No caso de desenvolvimento de aplicativos *applets*, o pacote import java.applet traz o suporte para o desenvolvimento.

A seguir, um exemplo simples do código de um *applet* ApptoAppl.java. Esse código deve ser armazenado em um arquivo de mesmo nome da classe que ele define, seguido da extensão**.java**.

```
//Copyright and License Sun Microsystems.

import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.swing.*;
import java.applet.Applet;
public class ApptoAppl extends Applet
implements ActionListener {
JLabel text;
JButton button;
JPanel panel;
private boolean _clickMeMode = true;
public void init(){
setLayout(new BorderLayout(1, 2));
setBackground(Color.white);
text = new JLabel("Este e um simples Programa!");
button = new JButton("Click Me");
button.addActionListener(this);
add("Center", text);
add("South", button);
}
public void start(){
System.out.println("Applet starting.");
}
public void stop(){
System.out.println("Applet stopping.");
}
public void destroy(){
System.out.println("Destroy method called.");
}
public void actionPerformed(ActionEvent event){
Object source = event.getSource();
if (_clickMeMode) {
text.setText("Botao Clicado");
button.setText("Click novamente");
_clickMeMode = false;
} else {
text.setText("Este e um simples programa");
button.setText("Click ");
_clickMeMode = true;
}
}
}

/*************************/
```

O pacote java.awt contém as classes relacionadas à interface gráfica, componentes de interações **WIMP**, janelas, ícones, caixas de texto, menus, etc. Contém ainda as classes para processamento de imagens e as classes que são empregadas no tratamento dos eventos gerados pelo componentes da interface gráfica (eventos do mouse, etc.).

Na figura 6 mostra-se o código HTML que carrega o *applet* no navegador. O *applet* é carregado e formatado dentro de uma página *Web* de uma maneira semelhante a uma imagem.

Na maioria dos navegadores, o arquivo HTML indica que uma imagem deve ser colocada na página. A imagem é carregada a partir do servidor W*eb* e mostrada no local apropriado: a imagem é desenhada dentro da janela do navegador.

Sempre que um *applet* necessitar carregar alguns dados de um arquivo que é especificado em suas funções uma URL relativa está associada, o *applet* normalmente usa seu código ou o documento para formar a URL completa (APPLETS, 2006).

As dimensões do *applet* na área do navegador são definidas pelas linhas das tags do arquivo HTML dentro da chamada ao *applet*, nos campos *width* (largura) e *height* (altura).

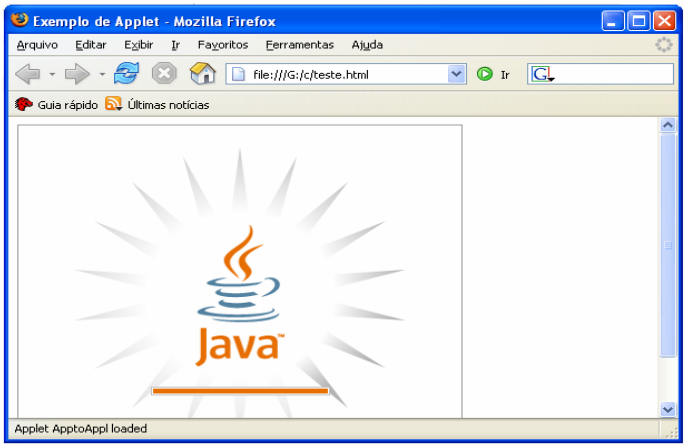

Figura 6 - Um Applet sendo carregado no navegador (APPLETS, 2006).

Após o carregamento do *applet*, começa-se, então, o serviço a que ele se destina. A figura 7 ilustra essa seqüência.

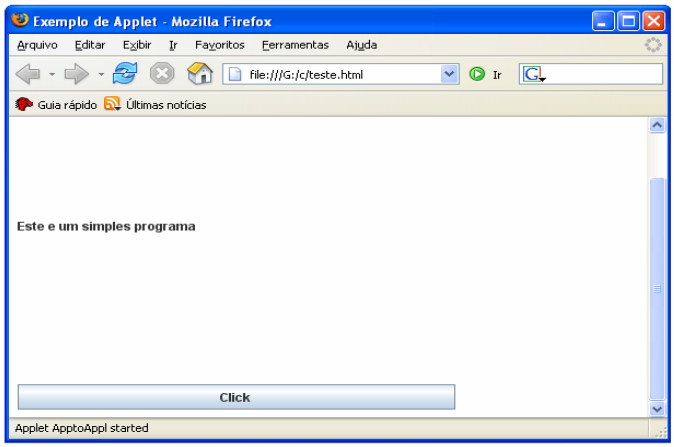

Figura 7 - Eventos do Applet (APPLETS, 2006).

2.5 Computação Distribuída usando a Linguagem C

A importância da linguagem C++ deve-se ao fato da mesma ser uma linguagem multi-paradigma, que suporta uma variedade de enfoques para o desenvolvimento de aplicações, incluindo: Programação estruturada; Abstração de dados; Programação OO; e, Programação genérica.

Para que se possa implementar objetos distribuídos na linguagem C++, deve-se criar primeiro instâncias desses objetos, e registrá-las para que possam cooperar entre si submetendo-os para o serviço principal.

Um *socket* é uma interface de comunicação bidirecional entre processos. E são representados como descritores de arquivos e que permitem a comunicação entre processos distintos na mesma máquina ou em máquinas heterogêneas, através de uma rede. Os s*ockets* são as bases das comunicações em redes TCP/IP e também são muito usados em comunicações entre processos no sistema operacional local (DUMAS, 1995).

Os clientes enviam pedidos através da referência a esses objetos, que são entidades que contêm as informações de comunicação utilizadas por s*ocket.*

Um processo cliente ao criar um s*ocket* tenta imediatamente conectar-se com o processo Servidor. Assim que a conexão é estabelecida, os processos começam a trocar informações. Qualquer um dos lados (cliente/servidor) pode encerrar a conexão, fechando o *socket*. Os serviços são descritos dessa forma:

- cliente invoca o *socket* e tenta conectar-se com um servidor;
- o servidor entra em modo escuta;
- o servidor aceita uma conexão;
- o cliente Envia dados;
- o servidor recebe e envia dados;
- cliente ou servidor fecha a conexão.

Este é o funcionamento básico de uma comunicação via *socket*. Não obstante, existem três tipos de comunicação de *sockets*:

- **datagram**: provê canais de comunicação bidirecionais sobre os quais pode-se enviar pacotes de dados a qualquer processo que tenha um canal definido. Não há garantia na entrega dos pacotes, pois o sistema implementa uma política de melhor esforço;

- **raw**: a comunicação provê acesso direto aos protocolos e interfaces de comunicação de baixo nível. É usada por processos que gerenciam ou monitoram a infra-estrutura de rede;

- **stream:** provê canais de comunicação bidirecionais ponto-a-ponto (entre dois processos pré-definidos) sobre os quais se pode enviar seqüências de bytes de qualquer tamanho, de maneira confiável.

Na linguagem C ou C++, várias bibliotecas oferecem suporte ao desenvolvimento de aplicativos para esse fim.

Na seção seguinte apresenta-se algumas considerações sobre este capítulo.

Excluído: mo

2.6 Considerações finais do capítulo

A evolução para a tecnologia dos componentes é progressiva para uma integração global e uma conseqüente necessidade de novas formas de transações entre os usuários dos serviços da *Web*, por exemplo, o comércio eletrônico tem direcionado novas arquiteturas tecnológicas para os sistemas distribuídos.

Essa evolução passa nomeadamente pela adoção da tecnologia orientada a componente com o recurso das infra-estruturas distribuídas baseadas em CORBA, DCOM, C++ e JAVA.

Uma arquitetura baseada em componentes facilita o desenvolvimento, de acordo com a necessidade**,** de novos módulos adicionais a partir de objetos existentes (componentes) que, suportados por uma infra-estrutura tecnológica distribuída, permite uma adequação natural do aplicativo.

Além de facilitar os processos de configuração e de inclusão de novas funcionalidades, a adoção das tecnologias de componentes associada à definição de interfaces padrão, possibilita a cooperação estreita com aplicações/módulos externos à própria solução. Também permite a coexistência, no mesmo aplicativo, de componentes provenientes de diferentes fornecedores e facilita, em cada caso, a união da melhor tecnologia que resolva o problema em questão (integração).

Os Serviços da *Web* criam um conjunto de funções empacotadas e distribuídas pela rede para que outros programas possam usá-las livremente. Outras arquiteturas como

CORBA, DCOM (*Distributed Component Object Model*), e as derivadas da linguagem Java (*Java Beans, Socket e Applets*) também possuem esse objetivo.

Andrew S. Tanenbaum (2003) faz comparações sobre os serviços das tecnologias CORBA e DCOM e conclui que ambas possuem diferentes estratégias para realizar a mesma função. Assim como em outras tecnologias concorrentes, CORBA e DCOM têm semelhanças e diferenças.

Nesse contexto, a linguagem Java disponibiliza ao desenvolvedor uma série de recursos, como o interpretador, o compilador, o *debugger*, o gerador de documentação, o visualizador de *applets,* o visualizador de *bytecode*, o compactador de classes entre outros.

A tecnologia disponibilizada pela linguagem Java dá suporte para diversos tipos de componentes: *Java Beans, applets e sockets*. Suas principais características são:

- a linguagem Java é *open source*;
- a programação é orientada a objeto;
- possui múltiplas heranças;
- API - interface de programação de aplicação está em constante evolução;
- os objetos Java possuem persistência;
- os objetos java possuem segurança;
- suporte total à portabilidade;
- usabilidade;
- integração com navegadores;
- persistência: devem ser preparados para serem serializados ou de-serializados tanto em tempo de projeto quanto em tempo de execução;
- gerador de assinaturas digitais;
- servidor de naming; no pacote org.omg.CosNaming.*;
- pode-se utilizar o padrão CORBA pelo pacote: org.omg.CORBA.*.

Concluindo esse capítulo, observa-se que para projetar aplicativos distribuídos deve-se considerar alguns problemas fundamentais apontados por Daniel A. Menascé (2002) que podem aumentar a latência da rede e, conseqüentemente, afetar a usabilidade. Além disso, há outros fatores que devem ser levados em consideração, tais como (MENASCÉ, 2002):

- a comunicação não confiável: os meios de transmissões podem apresentar deficiências como atenuação de sinais, ruídos térmicos, paradiafonia, colisões, etc;
- o acesso não autorizado às mensagens transmitidas pela rede e sua modificação afetando a segurança dos sistemas distribuídos;
- autenticação, autorização, auditoria e não repúdio;
- o custo da comunicação: largura de banda e alta latência, principalmente em aplicações multimídia.

O próximo capítulo apresenta os fundamentos que estabeleceram as inferfaces gráficas atuais baseadas em imagens *raster* e as interfaces fundamentadas em imagens vetoriais.

3. Fundamentação das Interfaces Gráficas GUI

Segundo Lauro F. Barbosa da Silveira (SILVEIRA, 2001), a semiose é a associação do sujeito do seu mundo interior com o mundo exterior, atribuindo-lhe uma idéia que ele identifica sob a sua percepção e memorização, ou seja, aquilo que está na sua recordação.

Essas relações existentes entre os símbolos, os ícones e os índices recebem um tratamento do mais alto grau de complexidade.

É essencial que um sistema computacional ofereça ao usuário um modo atrativo e convidativo, de forma a criar uma abstração do mundo do usuário no computador, ofertando uma familiaridade e uma cumplicidade entre a máquina e o usuário. Para que essa cumplicidade ocorra, o sistema deve ser baseado em metáforas do mundo real.

A concretização de uma comunicação entre pessoas só é possível quando o emissor da mensagem transmiti-la no mesmo código conhecido pelo receptor. Partes desse diálogo podem ser formatados por um ou vários signos. Assim que o receptor recebe a mensagem, ele formula uma idéia sobre o que o emissor quis dizer iniciando assim o processo da compreensão humana (JAKOBSON, 1970).

Um signo é algo que representa alguma coisa para alguém, por exemplo, uma palavra está associada a um símbolo (imagem) que contém o significante e o significado dos objetos. A disciplina responsável pelo estudo dos signos é a semiótica (MACHIRAJU, 1996).

No contexto deste capítulo o uso da metáfora desempenha uma função de tradução de um mundo digital representado pela linguagem do computador para a nossa linguagem natural. A este processo de conversão da linguagem do computador para a nossa linguagem, observa-se que a metáfora participa sob diversas formas.

O sistema semiótico de comunicação tem como fundamentação os processos envolvidos na produção e interpretação de signos. Para Pierce (PIERCE, 1997) o uso de metáforas como signos visuais levam em conta a relação existente do significado com seu interpretante, ou seja, a partir de certos ícones (às vezes meras formas) pode-se interpretá-los com a representação de um objeto.

Segundo Pierce (1997), no mundo moderno ainda existem diversos símbolos que representam objetos, ações e que induzem a uma reflexão de como usá-los. Porém, o que

parece simples para alguns, não é tão simples para outros (a maioria), dependendo da interpretação. Uma situação típica é quando a pessoa se depara com objetos conhecidos: ela sempre irá associá-lo com algo que já é de seu conhecimento. Outra situação é quando ela não sabe muito a respeito do seu significado.

Nessa abordagem, toda aplicação computacional é concebida como um ato de comunicação entre homem e o computador pelo uso de metáforas. Este processo recebe o nome de simbiose.

O uso da metáfora está comumente associado a um recurso literário, mas também é aplicado de propósito como uma ferramenta de comunicação a outros contextos comunicativos, notadamente para educação. Toda forma de educação é um processo de transmissão de uma nova informação a uma pessoa de maneira que ela possa assimilar e se relacionar com o que já é de seu conhecimento (GALLAGHER, 1978).

Na informática, o termo metáfora é utilizado como uma ferramenta que está centrada na interação do homem-computador e que sugestiona uma possível ação que ocasionará uma possível resposta da máquina.

3.1 Novas Metáforas: Interfaces e Tecnologia

As novas Metáforas significam a transposição do significado de um objeto para outro diferente. Por exemplo, o título deste parágrafo utiliza aqui a metáfora como um lugar onde estão implícitas todas as informações visuais (palavras, gráficos, etc.) sobre esse assunto. Uma definição de metáfora é apontada por Alan Frank Blackwell (BLACKWELL, 1998) como uma linguagem visual com a qual o usuário mantém diálogo com a máquina. A figura 8 ilustra a utilização do uso de metáforas a partir de sua abstração (signo).

Devido ao uso cotidiano de vários objetos e ícones, muitos conceitos foram incorporados pelo usuário leigo, sem saber do que se tratam, porém significam operações muito concretas em sistemas computacionais. A figura 8 ilustra essas operações.

Signo	Metáfora	Operação
	Copiar	Manipulação de Buffer
	Colar	Manipulação de Buffer
	Abrir Diretório	Criação de Processo
	Parar	Interrupção de Processo

Figura 8 - Utilização de Metáforas pelas GUIs, e significados em processos.

O modelo apresentado (paradigma) é composto por primitivas que regulam metáforas que o habitam e só é definido a partir de alguma atividade concreta. Em outras palavras, a concretização do uso de uma metáfora implica necessariamente em um paradigma, e este não é facilmente entendido sem o uso de uma metáfora (WINOGRAD, 1998).

No ano de 1945 Vanevar Bush (BUSH, 1945) publicou um artigo no jornal *The Atlantic Monthly* entitulado *"As We May Think"* que descreveu conceitualmente a máquina de Bush's *Menex*, que armazenava e recuperava informações de livros, registros, fotos e anotações. Nessa máquina a metáfora era utlizada nas abstrações da interface do usuário, botões, teclado e alavancas nela existente.

Um exemplo da aplicação desse modelo está na interface de uma aplicação, com o objetivo principal de mapear as ações do usuário em solicitações de processamento ao sistema (aplicação), assim como apresentar os resultados produzidos por ele (materialização do estímulo). O sistema ideal deve esconder a tecnologia do usuário.

O objetivo é deixar que as pessoas realizem suas atividades com o auxilio da tecnologia, e conseqüentemente, aumentem a sua produtividade cada vez mais por ser invisível, fora de vista, despercebida. Os usuários devem utilizar a macro função, não as variações implícitas na tecnologia. Um bom exemplo dessa tecnologia se encontra na urna eleitoral eletrônica, método utilizado no Brasil para a eleição dos representantes políticos. Nela, existem sinais do Código Braile, possibilitando o voto para os deficientes visuais, como também há imagens dos candidatos, que ajuda os analfabetos a elegerem seus representantes, por meio do reconhecimento da pessoa.

O objetivo final dos símbolos e dos ícones é tornar a comunicação homem/computador mais simples, isolando o usuário dos recursos computacionais complexos. No caso deste

projeto, os sistemas distribuídos possuem diversos níveis de abstração disponíveis a um usuário comum.

O modo de interação imediato para o homem são as linguagens naturais para os diversos idiomas, e para a máquina a sua linguagem primitiva que, depois dos primeiros códigos montadores, foi a linguagem por comandos.

Desse modo, deve-se analisar estes dois fluxos: o da linguagem natural para a de comando, e no sentido inverso, da linguagem de comandos (disponíveis para acessos aos recursos em diversos níveis da máquina) para a linguagem humana (na forma de ícones).

3.2 Simbiose homem/máquina e Interfaces de Usuário

A maioria dos desenvolvimentos da computação, geralmente, acontece primeiramente com o surgimento das idéias antes mesmo delas estarem disponíveis ou existirem em uma máquina. Existem diversos tipos de interfaces de conversação homem-máquina. A seguir, apresenta-se a seqüência do histórico das Interfaces de Usuários (UI), que contribuíram de forma acentuada para a computação moderna.

Excluído: Apresentaremos, a

Excluído: e um

A história das interfaces pode ser dividida em várias etapas de acordo com o estado da arte na época. Uma das primeiras concepções de interface de usuário foi a máquina de *Bush's Menex*, que conceitualmente expressava definições sobre as UIs, com o objetivo de resolver o problema de armazenamento/recuperação de informações. Ela possuía interfaces de entrada e saídas, mas nunca foi implementada (BUSH, 1945).

Um outro marco referencial foi o trabalho publicado no ano de 1960, intitulado de "*Man-Computer Symbiosis"*. Neste artigo, J.C.R. Licklider (LICKLIDER, 1960) externou "A simbiose do homem e o computador". Ele anteviu que o computador teria dispositivos de comunicação com o homem, e desse modo, a previsão de interface foi estabelecida. "*Em alguns anos os homens poderão se comunicar mais efetivamente com uma máquina face a face*". Mas para que ocorresse uma efetiva interação entre o homem-computador seria necessário que o homem pudesse desenhar gráficos, quadros, escrever notas e equações em uma superfície de exibição no computador.

É importante assinalar que nesta evolução histórica, o projeto "*Sketchpad*" TX-2 do MIT Instituto de Tecnologia de Massachussets *(Massachussets Institute of Technolology*), de

1963, apresentava uma característica revolucionária na computação: um ambiente de interação homem/computador - uma combinação de *software* e *hardware* que manipulava diretamente a interface, com o uso de uma caneta óptica (SUTHERLAND, 1963), (BOSHERNITSAN, 2004). O seu trabalho foi a base necessária para o desenvolvimento das Interfaces Gráficas de Usuário de hoje.

3.2.1 Linguagem Natural e por Comando

Esta seção trata-se das linguagens das formas de interações através da linguagem natural e da linguagem por comandos. As suas definições de interface devem ser entendidas como parte de um sistema computacional com o qual uma pessoa entra em contato físico, perceptiva e conceitualmente. Esta definição caracteriza uma perspectiva para a interface de usuário como se fosse um componente físico que o usuário observa e manipula, e outro conceitual, que o usuário interpreta, processa e raciocina. (MORAN, 1981).

Pierre Lévy (LÉVY, 1998) afirma em sua obra que o principal problema do diálogo com os computadores está nas diferenças entre as linguagens formais e as linguagens naturais utilizadas pelo homem.

Geralmente, a sintaxe e a semântica presentes nas linguagens de programações são descritas em uma Linguagem Natural. Tais definições são compreensíveis e acessíveis para a maioria dos usuários. O modo de interagir utilizando essa linguagem é mais atrativo para os usuários que dispõem de pouco ou quase nenhum conhecimento em informática. Entretanto, a Linguagem Natural não se aplica a todos os tipos de sistemas existentes.

Em uma determinada aplicação, o objetivo de usar a Linguagem Natural é aproximar o usuário da aplicação, privilegiando essa forma de diálogo.

Em algumas aplicações o usuário pode se expressar em uma Linguagem Natural, utilizando o seu próprio idioma com os semelhantes. Um exemplo de Interface de Usuário em Linguagem Natural é apontado no trabalho de Stephen Chong (CHONG et al., 2004) em reconhecimento de voz para diversos tipos de aplicações.

Para descrever essa Linguagem, necessita-se de um método formal que descreva a sintaxe e a semântica de maneira precisa e sem ambigüidades. A notação BNF *(Backus-Naur Form)* é um formalismo amplamente aceito para a definição formal da sintaxe. Por outro lado,

no campo da semântica, o assunto é mais complicado, pois a semântica é muito mais difícil de ser descrita que a sintaxe.

A linguagem de comando é a forma mais direta de um usuário se comunicar com o sistema operacional. Essa linguagem é oferecida por cada sistema operacional para que, por meio de comandos simples, o usuário possa ter acesso a rotinas especificas do sistema.

As interfaces baseadas em Linguagens de Comandos disponibilizam ao usuário possibilidades de interagir com o computador enviando instruções diretamente ao sistema por meio de comandos pré-determinados. Estas instruções podem ser compostas por um conjunto de teclas com funções pré-determinadas, por exemplo, a função da tecla CTRL exemplifica a linguagem de comando (TANENBAUM, 1995). Elas também podem ser feitas com um único caractere ou por abreviações curtas, mas sempre observando a sintaxe e a semântica da linguagem.

Uma das características desses sistemas é que o usuário tem que memorizar um grande número de instruções. Mesmo com os usuários mais experientes a sintaxe dos comandos da linguagem precisa ser relembrada e erros comuns de digitação podem ocorrer. A falta de uma padronização nos diversos Sistemas Operacionais é um fator importante na dificuldade de utilização na Linguagem Natural. Muitos dos sistemas existentes e atuais (Linux, FreeBsd, Windows, etc.) possuem interfaces do tipo Interface Linha de Comando (PREECE, 1994).

Excluído: ,

Excluído: (LINUX, 2005).¶

3.2.2 Bases da Teoria da Cognição e a Linguagem

A produção de uma tecnologia que auxilie os homens pressupõe a existência de um conhecimento anterior sobre eles e não somente das tecnologias disponíveis. A metodologia aplicada deve focar o processo pelo qual se possa adquirir conhecimentos que utilizem as teorias da compreensão, capacidades e limitações da mente dos usuários (USABILITY, 2002).

As Bases dessas teorias possuem fundamentações comuns nas áreas de psicologia cognitiva, ciência cognitiva e inteligência artificial que estudam o processo da transferência e absorção do conhecimento.

Para que se possa entender as maneiras de como estas bases são estabelecidas (recordação, aprendizado, interpretação e planejamento) no desenvolvimento de uma interface, os projetistas se apóiam em modelos cognitivos genéricos chamados de projeto de

sistemas centrado no usuário (*User Centered System Design – UCSD*), (USABILITY, 2002), (GULLIKSEN, 2003), (EMMUS, 2005).

3.2.3 Interface de Linha de Comando (CLI)

Desde os primórdios até os dias atuais o conceito de Interface por linha de comando é fundamental em sistemas computacionais. Esse tipo de interação continua presente na maioria dos sistemas operacionais atuais, porém esse ambiente apresenta uma rigidez de diálogo, onde o usuário interage com o computador através de comandos textuais baseados em uma determinada linguagem, o que exige um esforço cognitivo do usuário, portanto difícil para a nossa compreensão (CLI, 2006).

Utilizando-se as regras dessa interação, o computador conferirá a instrução digitada com as contidas em sua linguagem e efetuará a operação correspondente em uma seqüência lógica e coerente. Tendo em vista que o computador só entende instruções, elas deverão ser solicitadas passo a passo.

A interface fundamentada em linha de comando evoluiu de um grupo de usuários conectados a uma rede, com console, monitores de sistema como, por exemplo, no caso dos usuários do sistema operacional "MULTICS" (*MULTiplexed Information and Computing Service*) que tinham como modelo de interação uma série de transações de pedidos e respostas expressos em forma textual e os comandos possuíam um vocabulário especializado.

Nessas interfaces o *software* nos possíveis modos não era exploratório e nem interativo. Vejamos algumas características:

- a interface exigia uma carga de memorização do seu vocabulário de comandos;
- aprendizado exigia um maior esforço e um tempo longo no aprendizado;
- alta velocidade de processamento, com pouca utilização de memória na interface (pequeno espaço na tela do computador);
- rapidez de interação, para usuários experientes.

O sistema operacional MULTICS criado em 1965 foi um projeto de parcerias entre o MIT *(Massachussets Institute of Technolology*), o Laboratório Bell e a General Electric, com

o propósito de fornecer serviços de computação nos mesmos moldes da energia elétrica. Esse sistema operacional introduziu muitos conceitos inovadores na literatura da computação, mas a sua construção foi mais difícil do que se esperava. O Laboratório Bell saiu do projeto e a General Electric continuou sozinha (TANENBAUM, 2003).

O MULTICS executava suficientemente bem para ser usado em um ambiente de produção no MIT e em outros parcos lugares, mas falhou na idéia de um computador utilitário. Mesmo assim, o MULTICS teve uma enorme influência na evolução dos sistemas operacionais subseqüentes, como, por exemplo, os anéis de proteção, o sistema de arquivos estruturado em árvore, etc.

Na evolução desse sistema operacional, dois pesquisadores que trabalharam juntos no projeto do MULTICS, Ken Thompson e Dennis Ritchie, juntaram–se para reescrever o sistema em uma linguagem de alto nível chamada C. Tal sistema foi denominado **UNICS** *(UNiplexed Information and Computing Service)* e mais tarde foi trocado para UNIX (TANENBAUM, 2003).

3.3 Interface Gráfica de Usuário – (GUI)

Em 1968, um trabalho inicial chamado de NLS (Sistema *on-line*) foi a primeira implementação mais próxima do hipertexto atual. O NLS usou um novo dispositivo chamado *mouse* para facilitar a interação do homem com o computador, além de possuir uma interface de usuário que conectava o mouse a um ambiente de múltiplas janelas (*windowing*) provendo vídeo conferência (PARC, 2006).

O projeto Xerox Palo Alto foi um marco em interfaces gráficas em 1973. Embora não fosse um microcomputador, seus componentes de funcionamento ajustaram em uma mini torre de pequeno tamanho, tendo como característica mais notável o seu nodo de exibição (*display*) que era do mesmo tamanho e orientação igualmente a uma página de papel impressa, caracterizada em *raster*, gráficos de *bitmap*, com uma resolução de 606 por 808 (HORN, 2002), (BARDINI et al., 2002).

As Janelas poderiam ser alternadas independentemente de tempo em tempo, além de exibir apenas caracteres de texto fixo. Ao contrário do NLS, a parte gráfica da Xerox Palo Alto era baseada em vetor, podendo apenas exibir texto e linhas de reta. Além disso, possuía um teclado e uma versão modernizada do mouse com três botões. O próprio cursor do mouse

se tornou uma imagem *bitmap* (mapeada por bits), e pela primeira vez levou a forma de seta diagonal que apontava para alguma área da tela central (HORN, 2002), (BARDINI et al., 2002).

No início do projeto da Xerox Palo Alto, o gerenciador de arquivo exibia listas de diretório em duas colunas e era cercado por caixas, mas não havia nenhuma "janela". Existia um processador de textos, que podia exibir diversas fontes de letras, mas teve uma interface de usuário ligeiramente diferente com menus no fundo no topo da tela. Também havia um editor de gráficos de *bitmap*, mas teve sua própria interface de usuário diferente.

Os pesquisadores da PARC perceberam que era preciso uma interface de usuário consistente para as novas aplicações. Para que isso acontecesse em todo ambiente, era necessário inventar um novo código. A linguagem utilizada na época foi a Smalltalk (APPLE, 1992), (BOSHERNITSAN, 2004).

Esta era a primeira Interface Gráfica de Usuário moderna (vide figura 9), a qual foi fundamentada no Modelo de Visão e Controle (MVC) *(Model View Control)* original e deu origem às interfaces baseadas em janelas. A figura 10 ilustra o modelo MVC.

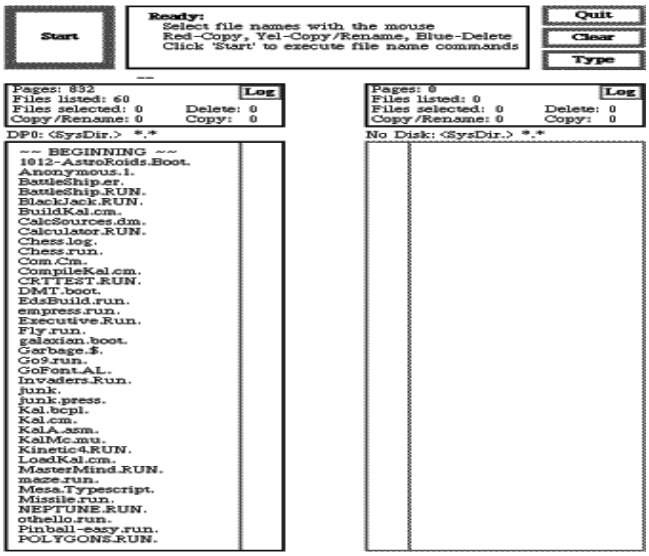

Figura 9 - Interface Gráfica Xerox Palo Alto (APPLE, 1992).

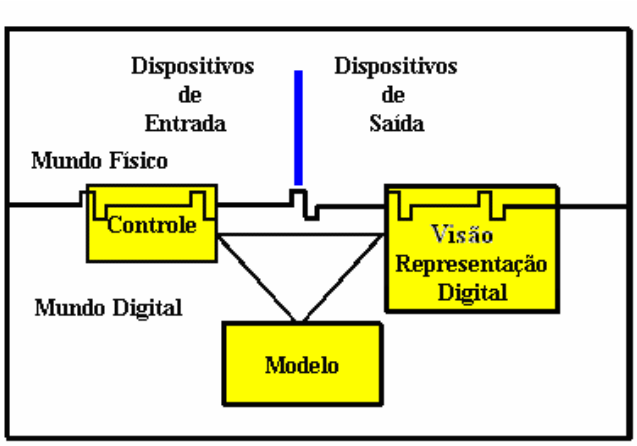

Figura 10 - Representação Gráfica do modelo de Interação - MVC (ULLMER, 2001).

A tríade de interação (usuário, *software* e *hardware*) tem que existir para que qualquer aplicativo seja útil. O modelo MVC original analisa os sistemas interativos em uma hierarquia de agentes. Um agente MVC consiste em um modelo que disponibiliza uma ou várias visões, de um ou de vários controladores. O modelo representa os núcleos funcionais do agente que podem representar alguns dados simples (inteiro, cadeia de caracter) ou de objetos que têm comportamentos complexos.

Uma decomposição desse modelo é explicada da seguinte forma: o Controle interpreta as ações do usuário provenientes da Visão e controla a execução das regras de negócio implícitas no Modelo, além disso, comanda a Visão para que ela apresente adequadamente a informação ao usuário.

O Modelo diz respeito ao gerenciamento da informação e ao comportamento da aplicação. Ele, em uma concepção simples, é uma mera representação do conteúdo do banco de dados.

A Visão não contém lógica de como funciona a aplicação, nem mesmo a lógica de navegação entre as diferentes interfaces apresentadas ao usuário. A Visão pode representar alguma lógica condicional que irá relacionar a sua forma de apresentação, como, por exemplo, as operações que o usuário pode realizar na interface gráfica (maximizando, reduzindo o tamanho ou escolhendo um determinado item do menu). Essas operações devem ser transformadas em eventos que são enviados para o Controle.

Existem vários exemplos da Visão para um modelo simples. Na visão destaca-se a utilização dos componentes gráficos *look and fell* [5] além do controle que recebe e interpreta os eventos do usuário, enquanto os retorna no modelo (modificação de estado) ou na visão (retorno instantâneo) (DRAGICEVIC, 2004).

Na próxima seção descreveremos os fundamentos dos componentes de interação gráfica WIMP.

3.4 Componentes de Interação WIMP

Os recursos utilizados na construção de interfaces de aplicativos utilizam uma série de componentes na forma de janelas, ícones, menus, e apontadores. A sigla WIMP é um termo que representa *Windows, Ícons, Menus and Pointers*. Sua invenção é creditada a Douglas Engelbart no Projeto Palo Alto (TANENBAUM, 2003).

Alguns autores afirmam que em um projeto de interface gráfica o alvo principal é a usabilidade centrada no usuário e a maior característica de um projeto de interface é o comportamento do humano na interação nos diferentes tipos de sistemas (PREECE, 1994), (MACHIRAJU, 1996). Este modo de interação permite a comunicação através de componentes gráficos virtuais denominados *widgets*. Este modo é implementado com o auxílio das tecnologias de desenvolvimento gráficas existentes (*Visual Tools*), por exemplo, as ferramentas TCL/TK e C++.

Uma interface gráfica é composta por diversos elementos visuais interativos. Para cada ambiente gráfico comumente existe uma ou várias bibliotecas desses componentes. Uma biblioteca define o formato dos padrões a serem utilizados na aparência e o comportamento da interface, chamado de veja e sinta (*look and fell*) de uma interface gráfica.

Atualmente, existem vários conjuntos de ferramentas (*toolkits)* utilizados na implementação das interfaces gráficas que disponibilizam o desenvolvimento dos formatos de janelas e que recebem ações através de mouse/teclado e de outros dispositivos.

Os *software* de construção de interfaces que implementam esses estilos de desenvolvimentos permitem a construção de ícones cuja função é receber a ação através do mouse ou do teclado, comportando-se como dispositivos virtuais de interação. O estilo WIMP não deve ser considerado como um único estilo de interação, mas a junção das

[5] *O termo look and fell é um estilo de interação chamado de veja e sinta.*

tecnologias de *software/hardware*, associadas aos conceitos de janelas e de controle (JANSEN, 1998).

Nas interfaces WIMP é possível encontrar os diferentes estilos de menus com manipulação direta, preenchimento de formulário, linguagem de comandos, etc. WIMP pode ser considerado um estilo ou um *framework* de interface apoiado pela tecnologia de interfaces gráficas GUI *(Graphical User Interface).*

O ambiente que utiliza esse componente ***Windowing*** possui uma área de exibição principal com uma barra de título e controles. Os controles, quando presente, podem ser usados para aumentar, minimizar, ou fechar uma janela. A barra de título geralmente pode ser usada para mover a janela de uma posição na tela. Algumas telas não possuem tal comportamento. **Barra de Rolagem** (*Scrollbars*) é o componente que permite a rolagem de textos na tela principal. Este componente pode ou não estar presente dependendo do estado que esteja a janela principal.

No cenário das interfaces visuais, os **ícones** (*icons*) são pequenos desenhos usados para representar uma entidade ou ações decorrentes de sua utilização. Representam uma metáfora do mundo real. Podem-se agregar em uma família ícones relacionados. Cada família é uma variante de um desígnio de ícone básico. Este componente possui dupla representação: a física (a imagem) e a lógica (o significado) (CHANG, 1990).

Os Menus (Lista de Opções), ou sub-tela, são apresentados na forma de múltiplas escolhas, ou seja, manipulação direta da interface. Existem diferentes tipos de Menus: *Pull Down, Pop-up, Radio buttons, Checkboxes, Dialog boxes*. Menu tornou-se um elemento padrão em interface WIMP. Um exemplo é mostrado na figura 11.

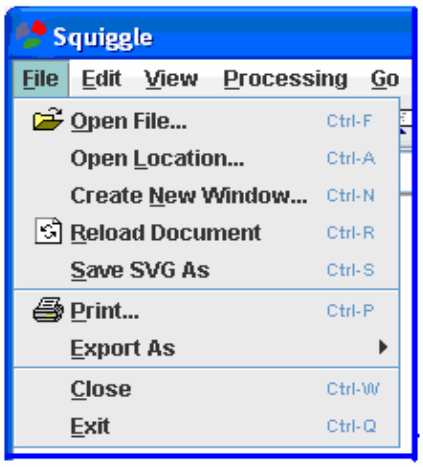

Figura 11 - Exemplo de Menu WIMP (BATIK, 2005).

O exemplo da figura 11 foi feito para o *software* BATIK (BATIK, 2005), aplicativo que possui diversas bibliotecas para trabalhar com o SVG. No contexto apresentado, uma caixa de diálogo é uma pequena janela com vários tipos de menus embutidos e, normalmente, alguns desses menus requerem preenchimento de campos. Eles são usados para fazer inserções de dados em aplicações possivelmente complexas (instalações).

A interação com os elementos WIMP só foi possível com o advento de dispositivos de entrada. Existem vários dispositivos neste grupo, sendo o mouse o mais comum. Ele interage com um ponteiro nas coordenadas gráficas da interface principal.

Credita-se à XEROX a maioria dos elementos gráficos que se utiliza como interfaces de WIMP (APPLE, 1992): uma interface de exibição mapeada por bits com diversos tipos de janelas, botões, menus, metáfora, uma arquitetura de *software* orientada a objeto e uma biblioteca de desenvolvimento. A linguagem de desenvolvimento usada para gerar esse ambiente foi a Smalltalk utilizada em 1970.

Muitos dos sistemas operacionais de hoje incluem tanto interface CLI como GUI. Por exemplo, o sistema Operacional Microsoft Windows possui uma Interface Gráfica de Usuário (GUI) com uma Interface Linha de Comando (CLI) "*command.com*" execução de *shell.*

Outros sistemas operacionais como Apple Mac OS X têm a interface GUI com um kernel UNIX e *shells* disponíveis. O SUN Solaris, FreeBSD e Linux possuem CLI com interfaces GUI opcionais. A maioria desses Sistemas Operacionais tem suas Interfaces

Gráfica de Usuário - GUI fundamentadas no padrão *MIT XWindows System* (X11R6) (Java Desktop, KDE), (GUITIMELINE, 2005).

Formatado: Espaçamento entre linhas: simples

3.5 Interface Tangível de Usuário (TUI)

Em 2001, Pierre Dragicevic (DRAGICEVIC, 2001) afirmou que a demanda por aplicações interativas estva tendo um maior destaque. Entretanto, os aplicativos, na sua maioria, ainda continuavam limitados ao uso dos dispositivos mouse e teclado. A complexidade de se apoiar dispositivos alternativos de interação em aplicações interativas é atualmente grande. Cada aplicação deve implementar um código específico para administrar cada dispositivo e todas as técnicas de interação desejadas por estes dispositivos. As melhores aplicações especializadas apóiam uma seleção limitada de dispositivos.

O objetivo das TUI é prover abstrações de mecanismos e de ferramentas que permitam desenvolver aplicações interativas que podem se adaptar a um método de utilização de inúmeros controles (ULLMER, 2001). As interfaces tangíveis de usuário se caracterizam pela junção física do mundo real e os controles para a informação digital, conforme ilustra a figura 12.

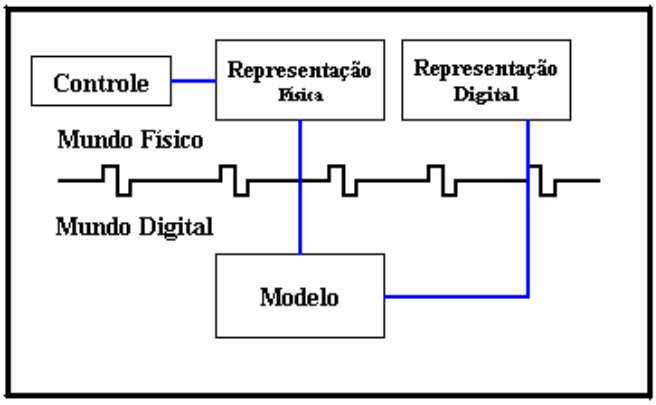

Figura 12 - Representação Gráfica do Modelo de Interação – TUI (ULLMER, 2001).

As interfaces tangíveis dão forma física à informação digital, enquanto empregam artefatos físicos ambos como representações e controles para mídia computacional. As TUI

juntam representações físicas, por exemplo: manipulação espacial de objetos físicos com representações digitais.

Pierre Dragicevic (DRAGICEVIC, 2004) conceituou as Interfaces Tangíveis de Usuário como sendo uma maneira diferente de controlar fisicamente os computadores modernos. Pierre afirmou também que, o poder de executar cálculos e a capacidade gráfica dos computadores domésticos vem crescendo a cada ano, porém essa evolução parece ter alcançado um nível de satisfação para os usuários acostumados a interagir apenas com um teclado e mouse, abrindo menus ou deslocando ícones com um mouse.

Um problema apontado pelo autor é que, nos dispositivos modernos, nada parece ter sido feito para se chegar a um acordo sobre os padrões desses fabricantes de dispositivos de interação.

Rodrigo García (GARCIA, 2006) em seu artigo, relatou a aplicabilidade dos componentes gráficos de interação em SVG no sistema SCADA *(Supervisory Control and Data Acquisition),* onde é disponibilizada uma interface gráfica necessária para monitorar e controlar processos de produção em instalações industrial.

Stephane Chatty (SVGMODELS, 2005) em seu trabalho afirma que a distância entre projeto de *Web* e projeto de Interface de Usuário está se estreitando a cada dia, como vem sendo demonstrado por recentes pesquisas em torno do conceito de aplicação da internet e que o SVG (*Scalable Vector Graphics)* está no centro dessa evolução, pois oferece uma alta definição de gráficos em duas dimensões. Este formato é familiar aos projetistas de interfaces gráficas de usuário com características de se ajustar dentro de um navegador de *Web* e aplicações *standalone* (sistema auto-suficiente, não necessitando de outro sistema para o seu funcionamento).

Nesse trabalho (SVGMODELS, 2005) o autor apresenta a construção de um simulador de vôo com painel sensível a todos os recursos de interação semelhantes aos componentes de interação WIMP. A figura 13 ilustra parte do simulador de vôo construído com SVG.

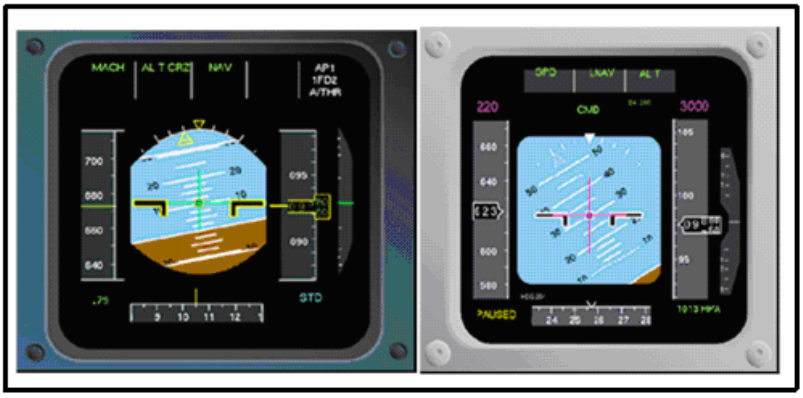

Figura 13 - Visualização em SVG de instrumentos de navegação aérea. (SVGMODELS, 2005)

A ferramenta utilizada no desenvolvimento do simulador de vôo foi a IntuiLab e o resultado do projeto foi uma interface de usuário altamente interativa - TUI que freqüentemente se caracteriza com capacidades de prover diferentes estilos de interação: reconhecimento de gestos, reconhecimento de fala, etc.

Ainda, segundo o autor, é difícil obter um protótipo avançado de interface em aplicações de multimodal que se ajuste em um navegador de *Web*. Uma conclusão apresentada é que a combinação do SVG com o código imperativo escrito em qualquer linguagem de programação (ou em outras linguagens de marcação) melhora a usabilidade dirigida ao *software*.

Aliado à exposição referenciada, observa-se que a crescente produção de dispositivos eletrônicos conseqüentemente demanda esforços cada vez maiores por parte dos profissionais, objetivando o desenvolvimento de *software* adequados a esses dispositivos: computadores pessoais, telefones celulares e pequenos computadores portáteis como os PDAs (*Personal Digital Assistant* ou assistente digital pessoal), que apresentam funcionalidades similares ao do computador de uso doméstico, por exemplo, calculadoras, editores de texto, correio eletrônico, agendas, navegadores e jogos. Entretanto, eles possuem diversas características que diferenciam uns dos outros, tais como: dispositivos de entrada e saída, capacidade de processamento, armazenamento e interface de usuário.

A seção seguinte descreve as principais características das imagens do tipo *raster* e vetoriais.

3.6 Interfaces Gráficas com SVG para Web

Uma interface gráfica de usuário possui uma coleção de elementos interativos. Estes elementos podem ser componentes geométricos, compostos de retas, curvas e outras formas geométricas combinadas; também podem possuir detalhes da aparência, como a cor, e suas variações; ou podem possuir informações para o interpretador sobre como interpretar uma determinada ação, ou formação da imagem ou de uma forma em particular.

As imagens presentes nas interfaces carregam consigo informações para o interpretador, normalmente, estas informações estão no início do arquivo da imagem e, é com elas que o interpretador vai realizar a interpretação.

Exemplos dessas informações para o interpretador são (WEBSTER, 1997):

- a sua extensão;
- o tipo de algoritmo de compressão;
- o número da versão;
- e uma lista dos elementos utilizados no arquivo.

A interface é tratada essencialmente por *pixel* a *pixel* e não a sua informação de contexto como a do formato vetorial, por exemplo. Uma imagem do tipo *raster* usa uma ordem de *pixels* com valores que indicam suas cores ou tonalidades, mapeamento de *pixels*. Essas imagens têm uma resolução fixa e não podem ser ampliadas sem perder a sua qualidade (distorção), fenômeno que não ocorre com as imagens vetoriais.

O padrão de gráfico baseado em *raster* inclui os formatos: BMP, JPG, GIF e PNG, entre outros. Outro padrão gráfico de imagem a ser comparado é o SVG, várias características incorporadas a esse padrão, e que dão suporte ao desenvolvimento de componentes de interação do tipo WIMP. As tabelas 2 e 3 mostram as principais características desses formatos (SVG, 2006).

Tabela 1 - Características dos diferentes formatos de imagens (WEBSTER,1997) (SVG, 2006).

Formatos	Nome Completo	Proprietário	Algoritmo de Compressão	Preenchimento de cores	Tipo
GIF	Graphics Interchange Format	CompuServe	LZW	1, 2, 3, 4, 5, 6, 7, 8	*Raster*
PNG	Portable Network Graphics	World Wide Web Consortium	Lossless/ DEFLATE	1, 2, 4, 8, 16, 24, 32, 48, 64	*Raster*
JPG	Joint Portable Graphics	World Wide Web Consortium	Lossy, DCT, RLE e Huffman	8-bit (escala de cinza), 12, 24 bit	*Raster*
BMP	Windows *bitmap*	Microsoft	Não, RLE	1, 4, 8, 16, 24, 32	*Raster*
SVG	Scalable Vector Graphics	World Wide Web Consortium	Não	24, 32	Vetor

Tabela 2 - Formatos que suportam animação na Web (WEBSTER, 1997) (SVG, 2006).

Formato	Animação	Camadas	Cores indexadas	Gerenciamento de cores	Transparência
GIF	Sim	Sim	Sim	Não	Sim
PNG	Não	Não	Não	Sim	Sim
JPG	Não	Não	Sim (modo 1-8 bits)	Sim	Não
BMP	Não	Não	Sim	Não	Sim
SVG	Sim	Sim	Não	Não	Sim

Conforme o exposto os gráficos *raster* são compostos por *pixels* e utilizam uma determinada técnica de compressão ou de compactação, essa técnica tem como objetivo reduzir o seu tamanho, para isso eles eliminam os *bytes* com redundâncias. Interessante observar que o formato SVG não utiliza nenhuma compressão, além de permitir cores em 24 e 32 bits com uma maior resolução.

Formatos de imagens vetoriais mais comuns incluem: AI - Ilustrador de Adobe, CDR - *CorelDraw*, CGM - Gráficos de Computador *Metafile*, SWF - *Shockwave* Flash, DXF - AutoCAD e SVG - *Scalable Vector Graphics*. Essas imagens possuem mais flexibilidade para

transformação e recomposição, mantendo a sua alta resolução sem distorção o que não ocorre no formato *raster*.

O vocabulário utilizado pelo padrão SVG foi proposto pelo W3C (*World Wide Web Consortium*) para representar gráficos bidimensionais, estáticos ou animados, em XML (*Extensible Markup Language*).

O alto nível das estruturas sintáticas e semânticas da linguagem XML somado às tecnologias como a do ECMA (*European Computer Manufactures Association*) e SMIL (*Synchronized Multimedia Integration Language*) vêm possibilitando não só sua utilização como linguagem alvo em sistemas de animação, mas também como linguagem fonte destinada à edição humana.

O SVG ao contrário de outros formatos de imagens utilizados na *Web*, seu arquivo não é binário, ele consiste em um documento no formato texto estruturado segundo as marcações definidas pela sua DTD (*Document Type Definition*).

Esse arquivo de texto é utilizado como arquivo de páginas na Internet e processado pelos programas de navegação. No caso do navegador Internet Explorer 6.0 é necessário a instalação de *plugin,* Adobe SVG *Viewer*. Já nos navegadores Firefox e Opera não é necessário o uso desse *plugin*, pois o código de interpretação é nativo.

Três tipos de categorias de objetos gráficos definidos pela DTD do SVG são (SVG, 2005): imagens vetoriais, imagens *raster* e texto, ou seja, pode-se utilizar os três elementos combinados em uma única aplicação.

A modelagem de imagens vetoriais, o principal enfoque desse componente, é realizada através da composição de elementos que descrevem estruturas geométricas primitivas, por exemplo: retângulos, círculos, elipses, linhas e polígonos.

Essa tecnologia promove o reuso dos objetos gráficos através de estruturas de definição e instanciamento. Essas estruturas, além de contribuírem para a criação de padrões gráficos, propiciam também a geração de arquivos menores. Características interessantes para um melhor desempenho de servidores e a latência da rede.

Os objetos gráficos gerados por essa tecnologia podem ser animados através de três abordagens (SVG, 2005): usando os elementos de animação do próprio vocabulário, usando o SVG DOM (*Document Object Model*) ou integrando o conteúdo com o SMIL que também foi desenvolvido pelo W3C, o SMIL é uma linguagem para integração de multimídia sincronizada que tem como objetivo a criação de apresentações multimídias na *Web*.

Os elementos de animação do SVG foram desenvolvidos pelo *Synchronized Multimedia Working Group* em colaboração com o W3C, criadores da especificação para

animações no SMIL. Com isso, exceto por algumas regras específicas do SVG, a definição normativa para seus elementos e atributos de animação seguem as especificações das animações em SMIL.

O SVG possui os seguintes elementos de animação que controlam tempo, variações de cor e sua transformação. Esses elementos descrevem as variações dos valores de atributos e propriedades dos elementos gráficos através do tempo.

O trecho do código seguinte exemplifica as primitivas de movimentação do SVG, onde se observa o controle do tempo que é determinado por atributos específicos nos elementos de animação. Entre os atributos de controle de tempo, estão os atributos que determinam o momento de início *(begin)*, duração *(dur=6s)*, término da animação *(end)* e o tipo de movimento (que nesse exemplo foi definido como linear).

```
<animateMotion values="15,43;40,30;65,30" begin="0s"
dur="6s"calcMode="linear"fill="freeze"/>
```

Várias ferramentas Open Source dão suporte ao desenvolvimento (construção e edição) de imagens vetoriais com o formato SVG, inclusive há ferramentas baseadas em Java. Exemplos dessas ferramentas são: Batik, Inkscape, Sketsa**,** svgl toolkit entre outras (CONVERSY et al., 2001).

Ressalta-se a existência e disponibilidade para plataformas Win32 e Unix. No caso das plataformas derivadas do Unix essas ferramentas estão contidas nos pacotes de instalação do FreeBSD e LINUX.

Uma caracteristica interessante das imagens SVG é que são mais leves e não perdem a qualidade ao serem redesenhadas (*Zoom*), conseqüentemente consumindo pouco esforço computacional na transmissão e na sua reconstituição (SVG, 2005).

Vários eventos podem ser adicionados à interface de usuário conforme se observa na tabela 3. Assim, o desenvovedor pode dar formas às imagens que sejam do seu interesse. Por exemplo, mudança de cor, o formato, o tamanho, indicando uma açao provocada pela interação do mouse, teclado ou outro dispositivo de entrada.

Excluído: 4

Tabela 3 - Eventos SVG (SVG, 2005).

GUI	Mouse	Estatus	DOM	Animação	Zoom/Pan
Focus In	Click	SVG load	Subtree Modified	Begin Event	SVG Resize
Focus Out	Mouse Down	SVG unload	Node Inserted	End Event	SVG Scroll
Activate	Mouse Up	SVG abort	Node Removed	Repeat Event	SVG Zoom
	Mouse Over	SVG error	Node Removed From Document		
	Mouse Move		Node Inserted IIIInto Document		

Além das características gerais aqui apresentadas, alguns testes (eventos semânticos do mouse) realizados com um ícone e um botão demonstraram a praticidade na interação usuário-computador fundamentada no modelo MVC semântico, conforme ilustra a figura 14.

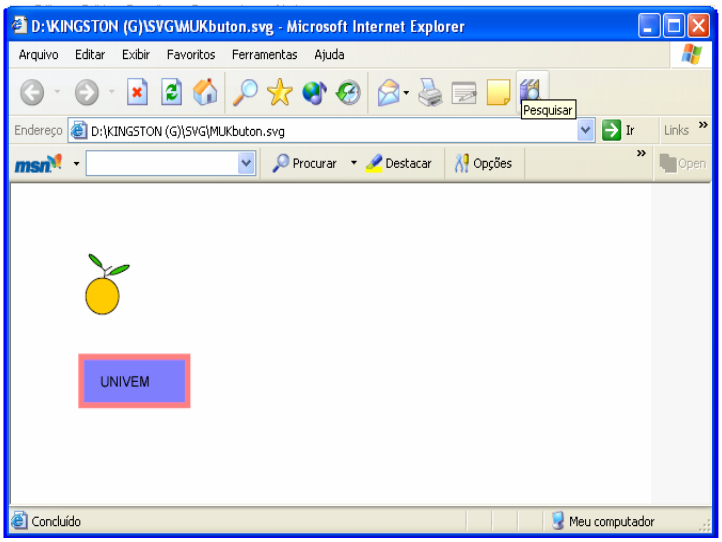

Figura 14 – Construção inicial de Ícones em SVG para o ambiente Windows (OLIVEIRA et al., 2006).

O trecho do código seguinte corresponde aos ícones da figura 14 e descreve as primitivas gráficas do formato SVG e as ações do botão retangular que acessa o link da UNIVEM e do botão circular programado com o evento de obter *download* (OLIVEIRA et al., 2006).

```
<?xml version='1.0' standalone='no'?>
<!DOCTYPE svg PUBLIC '-//W3C//DTD SVG 20001102//EN'
'http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-
20001102.dtd'>
<svg width='100%' height='100%'
xmlns='http://www.w3.org/2000/svg'
<rect id='BackDrop' x='-5%' y='-5%' width='110%' height='110%'
fill='none' pointer-events='all' />
   //desenha o retângulo
   <a
     xlink:href="http://galileu.fundanet.br/wdc/"
     target="_blank"
     id="a1549"
     transform="translate(1.81772,128.7923)">
    <rect width="80" x="18" height="20" y="-100"
style="opacity:0.5;fill:#0000ff;stroke:#ff0000;stroke-width:5"
id="rect1551" />
   </a>
 <a
link:href="ftp://mirror.aarnet.edu.au/pub/eudora/servers/unix/po
pper/"
     target="_blank"
     id="a1560"
     transform="translate(5.81772,128.7923)">
     <circle cx="40" cy="20" r="20" stroke="black"
     stroke-width="7" fill="yellow"/>//enchimento externo e
interno
 </a>
```

Nesse contexto outros componentes gráficos do tipo **WIMP** (*Windows, Icon, Menu and Pointers*) Janela, Ícones, Menus e Ponteiros foram desenvolvidos no decorrer desse estudo (vide tabela 4). Os códigos desses componentes estão no Apêndice G.

Formatado: Cor da fonte: Preto

Descrição dos componentes gráficos de interação em SVG desenvolvidos:

- Janelas: são componentes gráficos em forma retangulares, podendo ser redimensionadas conforme a conveniência do usuário;

Formatados: Marcadores e numeração

Tipos de caixas: caixas de lista; caixas de entrada; caixas de mensagem; caixas de diálogo.

- Ícones: são pequenas imagens que têm a função de abstrair uma macro função. Neste tipo de interface, os comandos são ações baseadas em uma analogia entre o cursor e a mão. O usuário interage com ícones, utilizando o mouse, teclado ou outro dispositivo

Formatados: Marcadores e numeração

equivalente por meio de ações. Por exemplo, clicar, arrastar (*drag-and-drop*), rolar sobre (*roll-over*), etc.

- Menus: são listas de sub-opções, sub-programas que oferecem uma comodidade ao usuário final. Nesses menus, a seleção de uma ou mais opções resulta em uma mudança no estado da interface. Existem diversas técnicas para se agrupar e apresentar as opções de menus. A mais comum é a categorização hierárquica das opções, na qual se deve tomar cuidado para selecionar nomes de grupos e de cada opção, para que reflitam as metas e tarefas do usuário. Esses componentes de interação virtuais são denominados *Widgets* e eles podem possuir comportamento complexo;
- Menus: são botões agupados: Pull Down, Pop-up, Rádio buttons, Checkboxes, Dialog boxes, e *Scrollbars.*
- Ponteiros: são formas de apontadores e podem assumir diversos formatos (setas, mão, dedo, cruz, ampulhetas, relógio, etc). Sua função é selecionar objetos, direcionar os eventos para marcar ou desmarcar os componentes de interação (coordenadas X e Y na tela).

Esses componentes gráficos de interação foram desenvolvidos utilizando as primitivas do SVG.

Tabela 4 - Componentes Desenvolvidos em SVG

W - Janela	**I** - Ícone	**M** - Menu	**P** - Ponteiro
Retangular com efeitos visuais	Circular com efeitos visuais	Retangular com efeitos visuais	Roll over Drag and Drop
Retangular sem efeitos visuais	Híbridos	Retangular com lados arredondados	Roll over Drag and Drop
			Wait/Copy

Outros eventos podem ser adicionados na Interface da Gráfica do Usuário, conforme relatado por Xiaohong Qiu (QIU, 2005) como, por exemplo, a aplicação de um conjunto de

Formatado: Fonte: Itálico
Excluído: *4*
Formatado: Fonte: Itálico
Formatado: Legenda, Manter com o próximo
Formatado: Cor da fonte: Preto
Formatado: Fonte: 12 pt, Cor da fonte: Preto
Formatado: Fonte: 12 pt, Cor da fonte: Preto
Formatado: Fonte: 12 pt, Cor da fonte: Preto, Inglês (E.U.A.)
Formatado: Fonte: 12 pt, Cor da fonte: Preto
Formatado: Inglês (E.U.A.)
Excluído: Nesse contexto, o

eventos do mouse e do teclado que permitem ao usuário gerar em tempo real ações altamente interativas.

Os eventos semânticos gerados pelo SVG representam a funcionalidade no domínio da aplicação, permitindo a esse modelo a reusabilidade máxima dos componentes na colaboração efetiva com interatividade da mídia nos diferentes conteúdos da *Web*, para diversos clientes e sobre diferentes ambientes de rede.

Além dessas características referenciadas, o autor (QIU, 2005) sugestiona uma interface uniforme, padrão para a próxima geração de clientes da *Web*, com acesso onipresente.

Três trabalhos de relevância mostraram a utilização dos componentes de interação gráficos em SVG. O primeiro de Rodrigo Garcia (GARCIA, 2006) no projeto SCADA, onde o autor demonstrou a utilização destes componentes na monitoração e controle de instalações em uma planta industrial geograficamente distribuída através de uma interface com componentes de interação vetorial. Já o segundo trabalho de Felipe Marinho (MARINHO et al., 2006) que referenciou o SCADA, demonstrou a utilização em diagramas sinópticos no âmbito da utilização do SVG na realidade do setor de automação de sistemas.

Nesse projeto (GENESys) o autor demonstrou a utilização do SVG em HTML com GUI, representando diagramas de operação sinópticos na *Web* e das soluções existentes no setor desse contexto. Ambos trabalhos demonstraram que a utilização de imagens do tipo *raster* na interação é pesada e ocupa por mais tempo o servidor no armazenamento/atualização, acarretando em atrasos nas respostas a qualquer solicitação realizada tanto em clientes *rich* ou *thiny*. O autor relatou os seguintes problemas com imagens *raster* (MARINHO et al., 2006):

> [...] A geração dinâmica de imagens "rasterizadas" (GIFs e PNGs) com animações é pesada e pouco eficiente;
> – Há necessidade de carregamento total da página a cada pedido de refrescamento. Uma vez que o conteúdo de uma página poderia incluir uma ou mais imagens dinâmicas de grandes dimensões, a quantidade de informação trocada entre clientes e servidor poderia ser muito grande e, por isso, condicionada pela largura de banda do canal de transmissão;
> – Há necessidade de uma taxa de refrescamento elevada. As características de tempo-real dos sistemas SCADA levam à necessidade de atualização freqüente da informação visualizada. Este factor, em conjunto com a eventual existência de outros clientes que efetuam acessos simultâneos ao sistema, introduz uma enorme sobrecarga no motor de geração de páginas que, por serem complexas e de grandes dimensões, pode conduzir ao atraso na disponibilizacão da resposta.

O terceiro trabalho foi o trabalho de Tazkiya (ALI, 2005), que utilizou o SVG com PHP no modelo MVC *(model/view/control)*, foi desenvolvido um protótipo de uma ferramenta de Engenharia de *Software* para testes com a especificação em linguagem Z (formal), usando os recursos de interações do SVG, esse aplicativo executa testes de caixas branca e preta, lado cliente e servidor (usabilidade).

Ainda, segundo a autora o uso combinado das tecnologias SVG e PHP implementado nesse estudo melhorou muito o tempo de resposta do servidor na *Web*. Algumas vantagens da escolha do SVG: suportes aos símbolos Unicode; animação; interação; provê uma boa visão gráfica (zoom in e zoom out).

Como limitação o trabalho destacou três pontos: o tamanho do plugin SVGview para download, a disponibilidade do código fonte de um arquivo SVG pode ser prejudicial a determinadas regras de negócios e a usuários, esses arquivos podem ser facilmente copiados, o uso de animações resulta em uma alta ocupação da CPU e que essa carga também depende do tamanho do arquivo e da complexidade da animação.

Excluído: Quebra de página

3.7 Considerações finais do Capítulo

Finalizando este capítulo, conclui-se que os diferentes formatos gráficos aqui apresentados possuem características heterogêneas. Por exemplo, utilização de diferentes algoritmos de compressão, entre outros.

As técnicas de compressão encontram sua utilidade na diminuição da quantidade de memória para armazená-la e na redução de seu tamanho para a otimização da largura da banda para a transmissão, porém agregam um maior custo de CPU (WEBSTER, 1997), conforme citado anteriormente o formato SVG não possui essa característica.

Essas características vão nortear os testes dessa dissertação com a finalidade de se obter dados sobre o impacto na arquitetura de *hardware* (processador/memória) durante suas requisições que é o foco dessa dissertação.

Assim o próximo capítulo dedicar-se-á análise arquitetural dos componentes do tipo WIMP em SVG.

4. Análise Arquitetural de Componentes Gráficos WIMP em SVG

A demanda pelos recursos na *Web*, está cada vez mais crescente, e torna-se necessário a adoção de novas tecnologias para o preenchimento de lacunas existentes. A imagem vetorial fundamentada em SVG sendo uma extensão do padrão XML poderá contribuir nesse intuito.

O objetivo deste trabalho é realizar um estudo sobre os componentes gráficos vetoriais de interação semânticos do tipo WIMP fundamentados na tecnologia SVG verificando o seu desempenho em uma arquitetura com baixo poder computacional, comparando com os demais formatos de imagens *raster*, enfatizando os aspectos como consumo médio de CPU e a utilização da memória.

O modelo M-MVC proposto por Xiaohong Qiu (QIU, 2005) decompõe uma aplicação de *Web* interativa que usa um modelo flexível e de fina granularidade, com evento natural de interações. Teoricamente, segundo o autor, qualquer parte de uma aplicação com interações de evento naturais pode unir um objeto ou organizar com acoplamento de mensagem.

Os eventos que representam as mudanças do estado que propagam ao longo da rede, como mensagens, e que interconectam um gráfico dinâmico de uma máquina de estado finito.

Na arquitetura de M-MVC (*Message-based Model-View-Controller*) o autor explorou a composição do modelo de visão (*View)* fundamentado em uma variedade de estratégias flexíveis destas fases. O importante neste modelo é que se abriu a possibilidade de interfaces gráficas e semânticas presentes no modelo MVC (QIU, 2005).

Este esquema provê muitas possibilidades para construir aplicações da *Web* com interface de cliente que facilite o acesso universal. O autor definiu o modelo em três fases: a da semântica, do alto nível de interface e a da visão, o termo *Port* significa a primitiva gráfica assumida pelo evento semântico, conforme ilustra a figura 15.

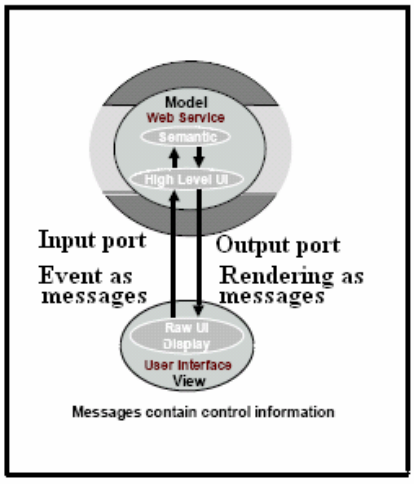

Figura 15 - Representação Gráfica do modelo M-MVC (QIU, 2005).

Em conformidade com o exposto, o conceito de projeto com eventos de Interface de Usuário inclui os eventos semânticos dentro do modelo de componente em um serviço da *Web*.

Esses eventos baseados em mensagens fazem o papel do Controle, unindo o Modelo e os componentes da Visão. As distinções principais entre os modelos MVC e M-MVC refletem as diferentes visões da composição de um sistema.

Esse trabalho irá comparar entre outros, o custo computacional dos diferentes formatos de imagens em relação ao formato SVG, que compõem a visão dentro do modelo MVC em uma aplicação cliente/servidora.

4.1 Definições das Métricas

O principal objetivo para definição das métricas é prover uma base de dados estatística para avaliação do desempenho (dados a serem coletados, a instrumentação utilizada na coleta destes, os dados e a forma de validação dos resultados) dos diferentes formatos gráficos presentes em uma aplicação *Web*.

Descrever também o comportamento do conjunto CPU - Cliente/Servidor, e utilização da memória quando submetidos a diferentes tipos de cargas (processos) geradas pelos diversos formatos de imagens durante execução dos testes considerados essenciais.

4.1.1 Métrica adotada - Tempos de Respostas

O resultado equivale à média dos tempos utilizados em relação ao número de requisições efetuadas, tendo a seguinte seqüência de atividades:

- avaliar a carga do processador entre a requisição e o recebimento de imagens padronizadas: 1024X768, nos formatos: BMP 16, BMP 256, BMP 24 bits, JPG, GIF, PNG e SVG; na situação: cliente/servidor;
- avaliar a carga do processador entre a requisição e o recebimento de um arquivo HTML vazio nos seguintes navegadores: Internet Explorer 6.0, Firefox 1.5 e Opera 9.02 no modo cliente/servidor;
- avaliar o percentual de carga do processador entre a requisição e o recebimento de um arquivo SVG embutido entre as *tags* de um arquivo HTML, utilizando o protocolo http nos navegadores: Internet Explorer 6.0, Firefox 1.5 e Opera 9.02. Nos testes foi usado o servidor *Web*: Apache Tomcat versão 5.0, com IP 192.168.0.1, a porta 8080 para acesso interno, e o IP 201.76.65.247 para acesso externo;
- avaliar o tempo de carregamento de páginas dos navegadores referenciados;
- avaliar o tempo de carregamento de páginas vazia navegadores referenciados;
- constatar, observar e registrar problemas efetivos de usabilidade durante os testes.

4.1.2 Captura da Carga do Processador

A partir do lançamento do processador Pentium, a arquitetura IA32, disponibilizou um contador de 64 bits, que mantém uma contagem precisa de todos os ciclos que acontecem no processador, que é incrementado em cada ciclo do relógio. Esse contador é iniciado com 0 quando é feito o *reset* do processador (IDL), sendo depois consecutivamente incrementado de acordo com o processo corrente. Este contador pode ser lido pela instrução RDTSC – *Read Time Stamp Counter*, que coloca os seus 32 bits mais significativos no registrador EDX e os 32 bits menos significativos no registrador EAX. (INTEL, 1997), (INTEL, 1998), (INTEL, 1999) (INTEL, 2002).

O número de ciclos decorridos durante a execução de um aplicativo pode ser obtido calculando o valor inicial (T1) e final (T2) do contador durante a execução de um processo. A diferença entre T2-T1 será igual ao número de ciclos da CPU utilizados pelo processo.

Este item é importante, pois é através da captura do *ticker* que se obtêm o percentual de ocupação da CPU no processamento (interpretação, espaço de memória no *buffer*) entre o teclar "*enter*" e a formação da imagem dentro do navegador. Esta captura foi feita através dos ciclos de pulsos alternados de sinais de tensão gerados pelos circuitos de relógio (*ticker*). Esse ciclo é delimitado pelo inicio da descida do sinal, neste caso é possível captar o tempo no nível de *hardware*, assim o tempo captado é mais preciso (principio da localidade). Por exemplo, o tempo de 1 segundo é igual a: ciclos/freqüência da CPU. Por exemplo, em uma CPU de 400Mhz. Têm-se: 400.000.000/400Mhz = 1s.

Para capturar a carga do processador foi necessário utilizar a instrução na linguagem *assembler*: RDTSC *(Read Time-Stamp Counter)* (INTEL, 1997), (INTEL, 1998), (INTEL, 1999) (INTEL, 2002) e depois convertê-las para uma estrutura de onde é calculado o percentual de ocupação, armazenando os valores em uma lista no *buffer* para depois escrevê-los em um arquivo de texto.

Usando a instrução RDTSC em um mesmo processo pode produzir resultados diferentes. Isto é causado freqüentemente pelos efeitos da memória cache. A primeira vez que se executa um processo, o mesmo pode consumir um determinado número de ciclos para encher o *cache* e transferir os dados para a memória *RAM*. Assim, ao usar o RDTSC repetidamente pode produzir resultados inferiores à primeira medida. Conseqüentemente, a média será inferior ao desempenho real do processador. Para evitar esse efeito, a cada medida coletada no desempenho do processador observado deve-se desligar e ligar a máquina novamente. Assim a média é a mais real possível, processo esse utilizado nessa dissertação.

Nesse trabalho foram efetuados vários testes com um *delay* na coleta do *frame* dentro do laço de: 100, 150, 200, 300, 600 e 1000 ms. Optou-se pela utilização do tempo de amostragem de 600 ms. Esse tempo demonstrou ser menos intrusivo na coleta dos dados, pois, quanto mais informação se recolhe mais se perturba, podendo alterar o comportamento de acessos no espaço e no tempo. Por exemplo, no nível dos *caches*, das páginas referenciadas.

Assim, a carga do processador foi obtida após realização das seguintes atividades:

- identificar a velocidade dos processadores nas máquinas envolvidas nos testes (CPUID) (INTEL, 1997), (INTEL, 1998), (INTEL, 1999), (INTEL, 2002);
- avaliar a utilização de processador do cliente durante as requisições. O resultado equivale à média de uso do processamento em relação ao número de requisições via conexões cliente/servidor;

- avaliar o consumo da utilização de processamento do servidor durante as requisições. O resultado equivale à média de uso do processamento em relação ao número de requisições via conexões.

4.1.3 Utilização da Memória

Com relação à utilização da memória, utilizou-se a ferramenta CurrProcess V1.11, a qual permitiu observar quatro parâmetros importantes na análise do uso de memória, a saber:

- o pico de utilização da memória durante as requisições;
- as faltas de páginas;
- o uso de arquivo de páginas;
- o pico de uso de arquivo de páginas. O resultado equivale à média do uso da memória em relação ao número de requisições solicitadas.

Essas avaliações foram importantes já que permitiram suprir as seguintes necessidades:

- possibilitou um melhor entendimento das imagens fundamentadas em *raster* e vetoriais SVG no ambiente da *Web;*
- verificou o impacto arquitetural das imagens binárias e vetoriais (SVG), entre outras.

4.2 Metodologia e Infra-estrutura Utilizada

Na realização dos testes foram utilizados dois algoritmos desenvolvidos em Visual C++ versão 6.0, um que verifica se o processador oferece suporte ao RDTSC, identifica o fabricante e a velocidade das máquinas envolvidas nos testes (CPUID), e o outro coleta os percentuais de ocupação do processador. Os códigos destes algoritmos estão no apêndice C e D.

Para identificar se o processador possui suporte ao RDTSC, e medir o consumo/ velocidade de processamento da CPU foram utilizados dois registradores (rdtsc 0fh/031h) e (cpuid 0fh/ 0a2h) conforme especificação da Intel RDTSC (INTEL, 1997), (INTEL, 1998), (INTEL, 1999) e (INTEL, 2002).

A escolha da utilização dos navegadores Internet Explorer 6.0, Firefox 1.5 e Opera 9.02 foi em função do suporte ao SVG. Em cada bateria de teste foi medida a carga gerada pela aquisição de uma página vazia (html vazio) para determinar seu custo separadamente. Foi medida a carga do processador entre o clicar e a execução total do processo. Foi medido também tempo decorrido no processo de solicitação e recebimento do arquivo (*downloaded*) do servidor até o cliente.

Após cada amostra testada, as máquinas cliente e servidor foram sempre desligadas a fim de se evitar que arquivos temporários fossem armazenados na memória secundária. Para isso foi utlizado o aplicativo Deep Freeze 2000XP (DEEPFREEZE, 2006) que mantém as características originais dos aplicativos instalados após a sua reinicialização.

As especificações do SVG testadas foram as 11.2 Specification W3C Working Draft 13 April 2005 (SVG, 2005), e a release SVG Mobile Test Suite 1.1 13 Dec. 2006 (SVG, 2006).

Nos testes desta dissertação foram adotados os seguintes procedimentos:

1. padronização das imagens 1024X768, nos formatos: BMP 16, BMP 256, BMP 24, JPG, GIF, PNG e SVG;
2. avaliar a latência da rede no processo de recuperação de um documento *Web*: HTML;
3. caracterizar a largura de banda da rede;
4. caracterizar a velocidade dos computadores cliente/servidor, processadores, capacidade de memória RAM e caches, etc;
5. enviar uma requisição do cliente aos componentes observados no servidor;
6. registrar e organizar os resultados;
7. reiniciar as máquinas cliente/servidor;
8. repetir novamente o quinto passo até a finalização dos testes.

Os testes foram realizados conforme a indicação anterior, sendo comparados os diversos formatos de imagens do tipo *raster* com o formato SVG, padronizados com as seguintes dimensões 1024x768. Para os testes internos, o servidor possuia o IP 192.168.0.1 com a porta 8080 liberada, e a máquina cliente estava configurada com o sistema operacional Windows Xp.

Foram realizados os seguintes testes observando-se as seguintes métricas: consumo da CPU, ocupação de memória, falta de página, pico de uso do arquivo de página e o tempo de

carregamento da página no lado cliente. No lado servidor foi medida a carga relativa a cada processo.

As imagens foram embutidas dentro das *tags* do html juntamente com um *script* para medir o tempo de carregamento da página no lado cliente. Cada imagem possui seu respectivo arquivo html e esses arquivos foram colocados no diretório http//192.168.0.1:8080/webdav/imagens/, do servidor *Web* Tomcat 5.0.

Na execução dos testes, foram utilizados os seguintes equipamentos:

Formatado: Fonte: Itálico

Excluído: , usando a ferramenta AIDA32

No Cliente:

Identificação Micro Computador Local – House

Formatado

Sistema operacional: Microsoft Windows XP Professional 5.1.2600 (WinXP Retail)
Internet Explorer versão: 6.0.2800.1106 (IE 6.0 SP1)
Tipo de processador, AMD K6-2 500 Mhz
Memória: 316 MB. Tempo de Acesso: 70ns. Cache: L1 – 32Kb. L2 – 0
Chipset da Placa Mãe: SIS 530.
Disco rígido: SAMSUNG SV0221N 20 Gb - 5400 RPM.
Partições: C: 20 GB.
Adaptador Gráfico: SIS 530/620 (4MB). Acelerador 3D SIS 86C306.
Rede: endereço IP principal: 192.168.0.94.
Adaptador de Rede: Realtek RTL8029(AS), 10Mbits PCI Ethernet Adapter.

No Servidor:

Identificação Micro Computador – HOUSE-MESTRADO.
Sistema operacional: Microsoft Windows XP Professional 5.1.2600 (WinXP Retail).
Internet Explorer versão: 6.0.2800.1106 (IE 6.0 SP1).
Tipo de processador: Intel Pentium II, 400 MHz (4 x 100).
Memória: 256 Mb. Tempo de Acesso: 70ns. Cache: L1 – 16Kb. L2 – 512Kb.

Formatado: Alemão (Alemanha)

Chipset da Placa Mãe: VIA VT82C693 Apollo Pro Plus.
Disco Rígido QUANTUM FIREBALLlct20 10 (10 GB, 4500 RPM, Ultra-ATA/100)
Adaptador gráfico SiS 5598/6326 (8 MB), Acelerador 3D SiS 86C326.
Rede: endereços IP: 192.168.0.1/201.76.65.247.
Adaptador de Rede: Realtek RTL8139, 100 Mbits, PCI Ethernet Adapter. IP interno (192.168.0.1).
Adaptador de Rede: Realtek RTL8139, 100 Mbits, PCI Ethernet Adapter. IP externo (201.76.65.247).

Hub:

Identificação: Encore 8 portas. 10/100 Mbits. Mod. ENH908-NWY

Dados obtidos utilizando a ferramenta AIDA32.

Formatado: Normal

4.3 Ocupação do Processador

Formatado: Fonte: 14 pt, Negrito

No decorrer deste trabalho, com a realização de vários testes observou-se a ocupação do processador, memória e o tempo de aquisição dos arquivos com formatos *raster* e vetorial embutidos entre as tags de um arquivo html. As tabelas 5 e 6 descrevem os *software* utilizados no cenário de testes. As seções seguintes apresentam os resultados obtidos juntamente com as respectivas análises.

Excluído: 5
Excluído: 4
Excluído: 5
Inserido: 5
Inserido: 4

Tabela 5 - Características dos software Utilizados

Cliente	***Software***	**Versão**	**Tamanho**
	Inkscape	0.44	
	Windows XP	Internet Explorer 6.0 Firefox 1.5 Opera 9.02	38.040 Mb. Incluso o *plugin* do SVG 29.187 Mb. 18.886 Mb
Servidor	Windows XP	Internet Explorer 6.0	
	Java	j2sdk1.4.2_07	
	Servidor Web	Tomcat 5.0	

Excluído: 6
Inserido: 6
Excluído: 5
Inserido: 5
Excluído: 6

Tabela 6 - Ferramentas de monitoração

Ferramentas de Monitoração	**Desenvolvedor**	**Versão**
CurrProcess	Nir Sofer	1.11
Aida 32	Tamas Nikos	3.93
Process Explorer	Sysinternals	10.2

4.4 Resultados Obtidos

4.4.1 Usando o Navegador Internet Explorer 6.0

4.4.1.1 Consumo de CPU

O comportamento do conjunto CPU/Cache com essas cargas geraram resultados que podem ser comparados. Na execução de cada processo (carga) a CPU executa uma série de operações de cargas naturais no sistema, como *jumps*, chamadas de funções e procedimentos, operações aritméticas, etc.

Após a execução de cada teste, foram coletados os seguintes dados de cada amostra: pico máximo de ocupação/percentual médio de ocupação do processador, tempo médio de aquisição, pico de ocupação da memória, falta de página, uso do arquivo de página e pico de uso do arquivo de página. Os dados são representados na tabela 11 do apêndice B.

Os ítens contidos nessa tabela representam as médias de 30 amostras efetuadas para cada arquivo analisado. Os valores obtidos demonstram o consumo de CPU nos diferentes formatos que possuem como características semelhantes: a cor e o tamanho da imagem. O formato SVG wimp (Retângulo) possui apenas uma primitiva gráfica (um vetor) que será a referência (medida de interesse) com a forma wimp (Círculos) que contém 19 primitivas gráficas. Os resultados do wimp Retângulo são visualizados na figura 16.

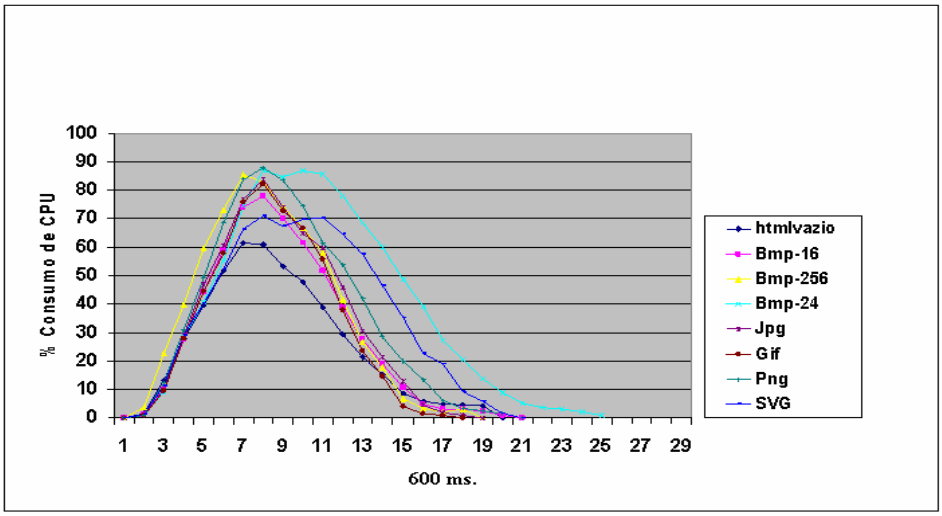

Figura 16 – Ocupação da CPU com o Navegador Internet Explorer 6.0. – Retângulo.

Analisando os resultados com relação ao consumo de CPU, observa-se que:

1. o formato SVG obteve o menor percentual de pico máximo de consumo da CPU (70,29 %). Já o consumo médio obtido foi de 40,36%, com um tempo médio de aquisição de 2,94 seg. (vide tabela 11) semelhante ao do formato Bmp-24, comprovando que mesmo sendo um arquivo gravado em ASCII (XML) de apenas 1Kb, que no processo de recuperação da imagem (codificar/decodificar/renderizar) é transformado em uma imagem com resolução de 24/32 bits;
2. no tocante ao consumo de CPU no lado servidor destaca-se que a menor taxa de ocupação do processador foi obtida com o SVG (1%). Já o formato Bmp-24 ocupou a maior média do processador (8,7%) com pico máximo de (10,16%) vide figura 17 que ilustra o processo de servir de acordo com as regras de negociação imposta pelo parser do navegador cliente;
3. outro ponto de destaque com o formato Bmp-24 (2,305 Mb.), o mesmo ocupou por mais tempo o processador, durante a aquisição (2,94) segundos, com um consumo médio de processamento na ordem de (44,31%), e pico máximo de (87 %);
4. em relação aos valores obtidos com o formato Png que gerou uma carga com um valor (41,12%) de consumo médio da CPU durante o tempo de aquisição (1,49 seg.), obtendo o maior pico máximo de ocupação do processador (87,64%);
5. o formato Gif (que visualiza apenas 8 bits) obteve o menor custo de processamento em relação aos demais formatos. Já o formato JPG obteve o memor tempo médio de aquisição: 0,58 segundos (vide figura 22).

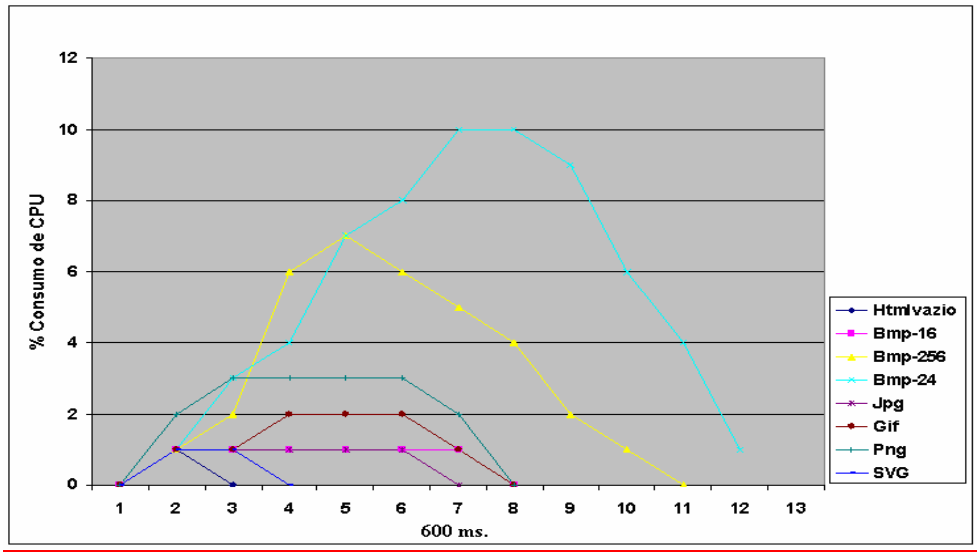

Figura 17 – Ocupação da CPU Lado Servidor com Cliente

Na próxima seqüência de testes são apresentados os resultados contidos na tabela 12 do apêndice B, os ítens coletados representam as médias das amostras efetuadas para cada arquivo analisado do wimp Círculo.

Essas imagens em relação aos formatos wimp retangulares que são preenchidos totalmente com uma única cor o formato Círculo apresenta uma maior complexidade, com gradientes de cores que variam de 2 a 24.000 bits, com pouca repetição de cores e com espaços vazios. A sua visualização é suportada apenas pelos formatos Bmp-24, Png, JPG e SVG.

Esse formato foi criado através do aplicativo Inkscape versão 0.44 (vide figura 18) e como característica básica ela composta de 19 vetores. A figura 18 ilustra esse WIMP.

Figura 18 – Forma Circular em SVG para Navegar.

Na próxima seqüência somente o formato SVG será comparado com os valores obtidos com a sua referência (wimp Retângulo) que contém apenas um elemento vetorial. A figura 19 ilustra a resposta do processador submetido a essa nova carga.

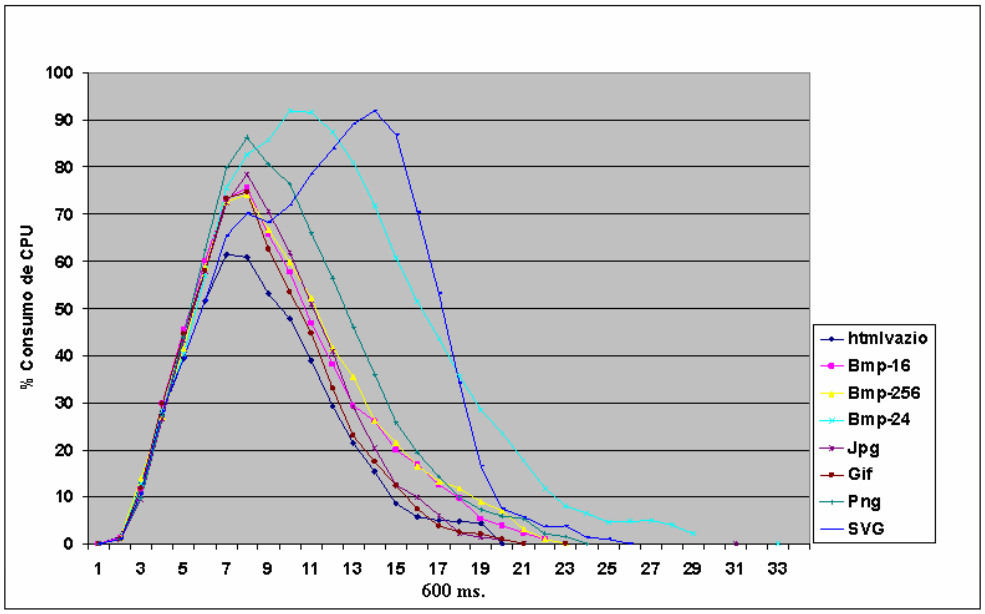

Figura 19 - Ocupação da CPU com o Navegador Internet Explorer 6.0. – Círculos.

Analisando os dados obtidos e mostrados na tabela 12 (ver apêndice B) têm-se:

1. com excessão do formato Bmp-24 que obteve como resultado 92% de pico máximo de utilização da CPU, 46,45% de consumo médio de CPU e com o tempo médio de aquisição de 4,29 segundos o SVG nessa série ocupou mais o processador 50,81% e um pico máximo de 92,13% e o tempo de aquisição de (3,23) segundos em relação aos demais formatos que possuem tamanhos de arquivos maiores, caracterizando um alto custo computacional no lado cliente;
2. analisando os dados de consumo de CPU no lado servidor (figura 20 que ilustra o processo de servir de acordo com as regras de negociação imposta pelo parser do navegador cliente) observa-se que o menor indíce foi obtido pelo formato SVG (1%). Já o formato Bmp-24 que obteve 9,63% de ocupação.

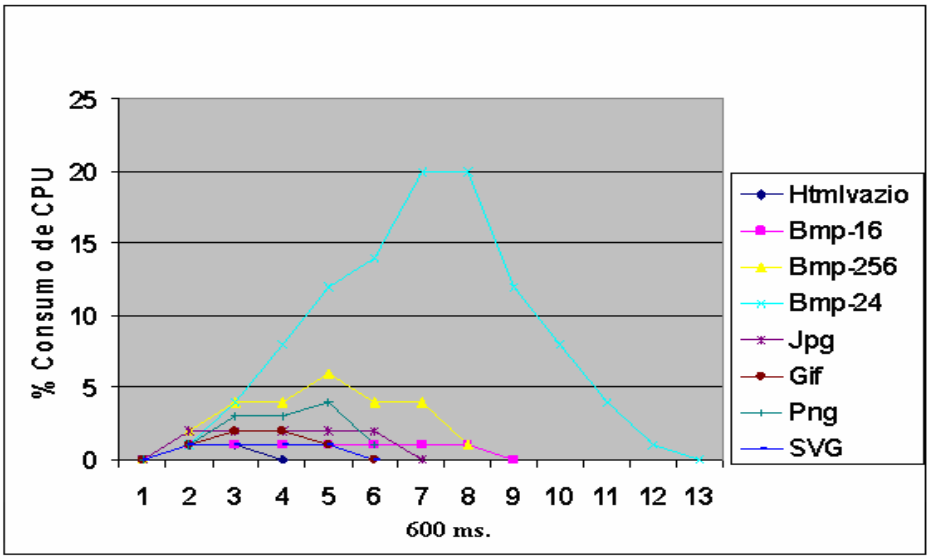

Figura 20 – Ocupação da CPU Lado Servidor – Círculos.

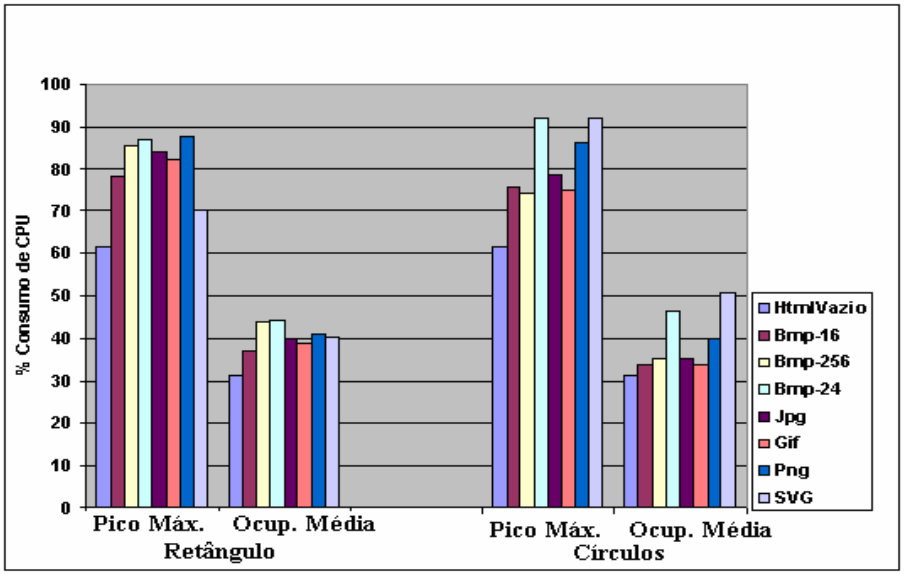

Figura 21 – Pico Máximo e Ocupação média de CPU – Retângulo/Círculos.

Comparando os dois resultados WIMP (Retângulo/Círculos) nesse navegador têm-se que:

1. o wimp SVG Retângulo obteve o melhor desempenho, que o torna interessante

para o desenvolvimento de botões, ícones e simples menus;

2. na evolução de 1 para 19 vetores, observa-se um acréscimo de 21,84% de pico máximo de utilização, 10,45% de consumo médio de CPU (figura 21) e apenas um acréscimo de (0,29) segundos na aquisição. Vide figura 22;
3. uma outra característica importante observada nessa seqüência de testes é que existe uma relação entre o número de elementos vetoriais contidos no SVG (Retângulo/Círculos) com o custo do processamento (codificar/decodificar/renderizar);
4. o formato Png obteve um alto consumo de processamento, com pico máximo de 86,38%, consumo médio de 40,02% e tempo médio de aquisição de 2,15 segundos;
5. no lado servidor, a menor taxa de ocupação do processador foi obtida com o SVG Retângulo/Círculos (vide figura 23).

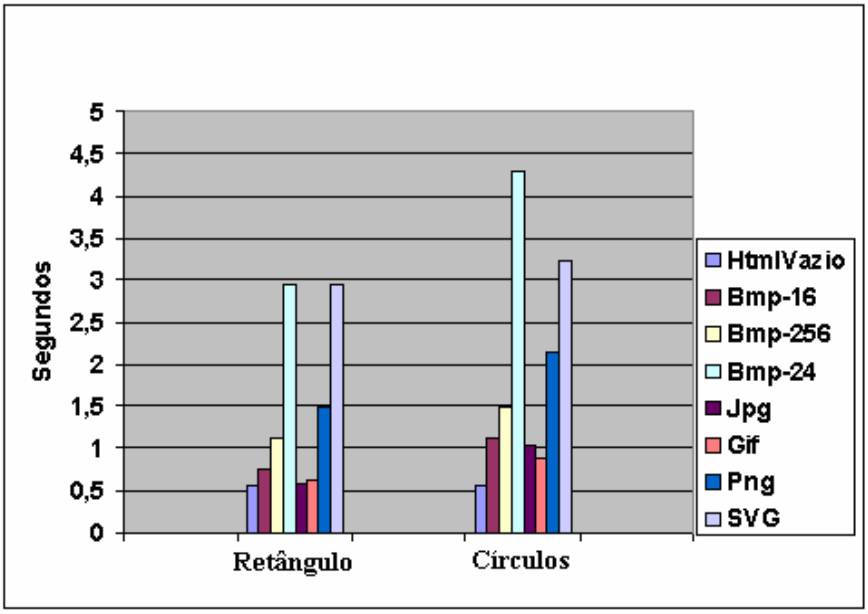

Figura 22 - Tempo de Carregamento das páginas - Retângulo/Circulos.

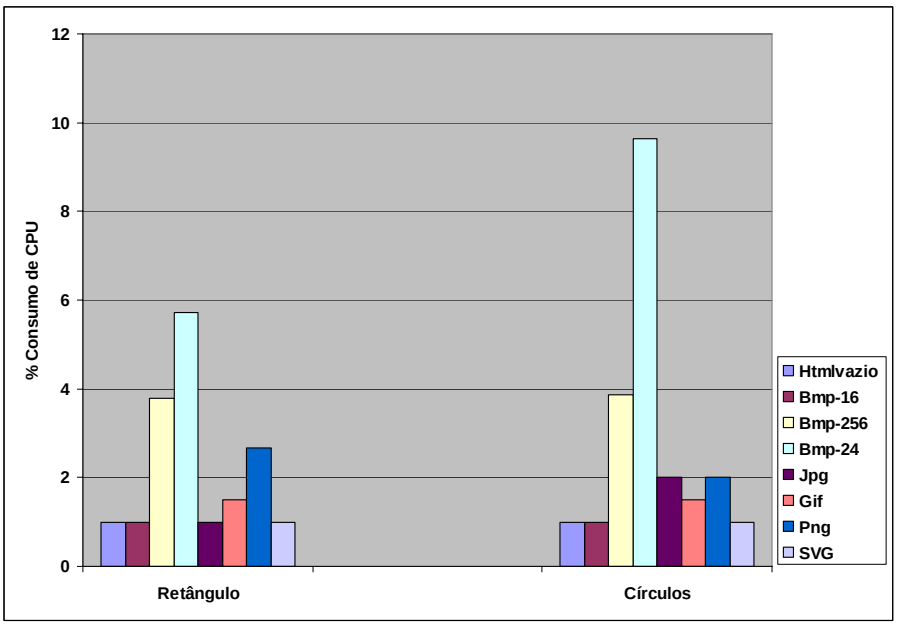

Figura 23 - Consumo CPU Servidor – Retângulo/Círculos.

4.4.1.2 Gerenciamento de Memória

No que tange ao Gerenciamento de Memória, o SVG obteve os maiores indices de ocupação de memória (25.648K), maior uso do arquivo de página (15.532K) e um alto índice de falta de página (7489) em relação aos demais formatos analisados. Os índices de falta de página são proporcionais aos índices de ocupação da memória pelo arquivo e essa relação se manteve em todos os formatos testados no navegador Internet Explorer 6.0 (Círculos). A figura 24 mostra esses dados.

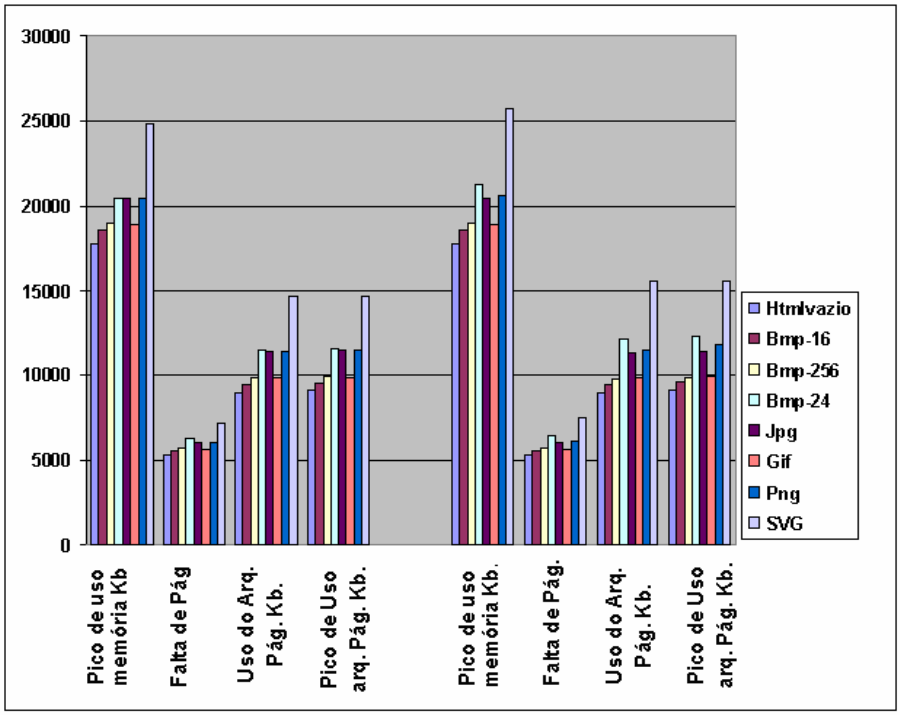

Figura 24 - Gerenciamento de Memória – Cliente Internet Explorer 6.0 - Retângulo/Círculos.

Na tabela 7 compara-se os dados estatísticos do SVG de 1 e de 19 primitivas gráficas nesse navegador. Conclui-se que:

1. com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,9221 para o consumo médio de 40,36% e um nível de confiabilidade de 97,71% para o wimp Retângulo;
2. em relação ao tempo médio de aquisição de 2,947 seg. (wimp Retângulo), com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,0422 seg. e um nível de confiabilidade de 98,56%;
3. com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 1,2384 para o consumo médio de 50,816% e um nível de confiabilidade de 97,56% para o wimp Círculos;
4. em relação ao tempo médio de aquisição de 3,233 seg., (wimp Círculos), com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,04902 seg. e um nível de confiabilidade de 98,48%;

Excluído: 7
Inserido: 7
Excluído: 6
Inserido: 6
Excluído: 7

Tabela 7 – Desempenho do Wimp SVG - Internet Explorer.

Estatísticas	wimp Retângulo	wimp Círculos
Média CPU	40,36%	50,81%
Desvio Padrão	2,577	3,461
Variância	6,645	11,981
Intervalo de Confiança α = 0,05	± 0,9221	± 1,2384
Nível de confiança	97,71%	97,56%
Tempo Médio	2,947 seg.	3,233 seg.
Desvio Padrão	0,118	0,137
Variância	0,013	0,019
Intervalo de Confiança α = 0,05	± 0,0422	± 0,04902
Nível de confiança	98,56%	98,48%

4.4.2 Usando o Navegador Firefox 1.5

Na próxima sequência de testes mostrados nas (tabelas 13 e 14 do apêndice B) os valores (visualizados na figura 25 e 27) representam as médias obtidas para cada arquivo analisado Retângulo/Círculos. Esses valores são as referências com relação ao navegador Firefox 1.5. Importante, lembrar que o objetivo não é comparar os navegadores, e sim os diferentes formato de imagens quando usados em aplicações WEB.

4.4.2.1 Consumo da CPU

Na próxima sequência de testes, inicialmente se verifica o comportamento de componentes WIMP do tipo retangulo, e depois se usa o formato Círculos, apresentando algumas comparações com os formatos testados principalmente com o SVG, porém usando como base para os experimentos o navegador Firefox.

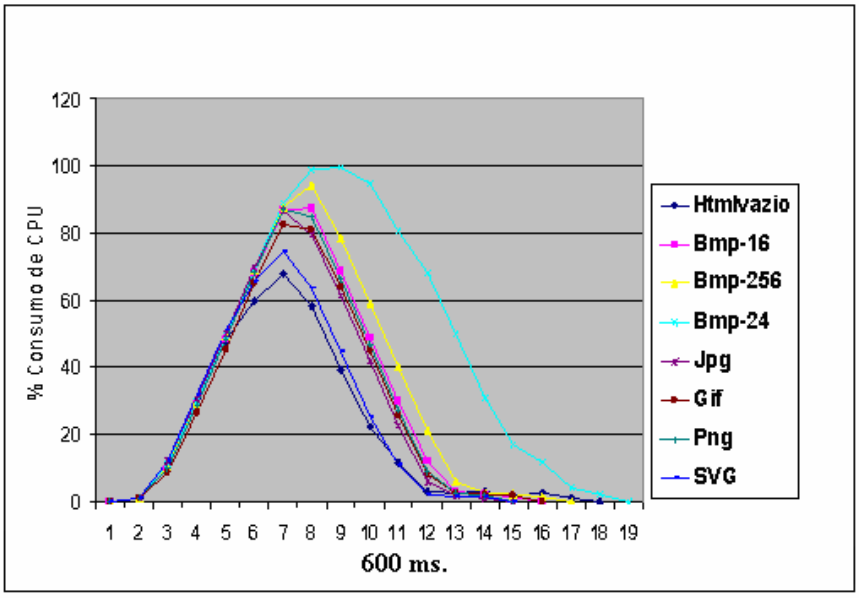

Figura 25 – Ocupação da CPU com o Navegador Firefox 1.5 – Retângulo.

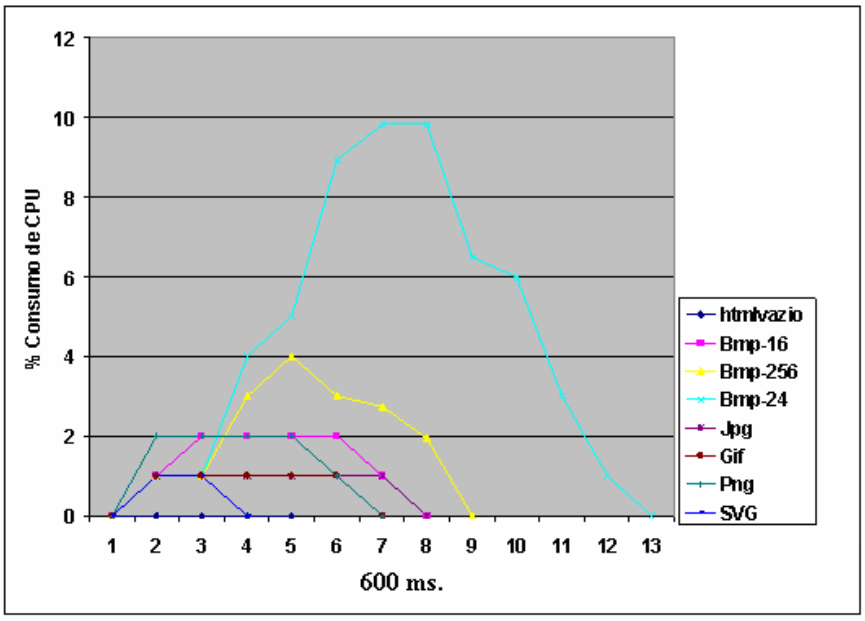

Figura 26 - Consumo CPU Lado Servidor com Cliente Firefox 1.5 – Retângulo.

Na próxima sequência de testes com o formato Círculos apresenta-se algumas

comparações com os formatos testados principalmente com o SVG, porém usando como base para os experimentos o navegador Firefox.

Em relação aos demais formatos, o SVG nessa série, ocupou o maior percentual de ocupação do processador, (100%) de Pico máximo e (64,46%) de consumo médio. Conclui-se que nesse navegador o *Parser* é mais rígido na análise dos 19 vetores. (vide figura 27). Na ocupação do servidor o formato SVG (vide figura 28 que ilustra as caraterísiticas no processo de servir de acordo com as regras de negociação imposta pelo *parser* do navegador cliente) ocupou em 1%.

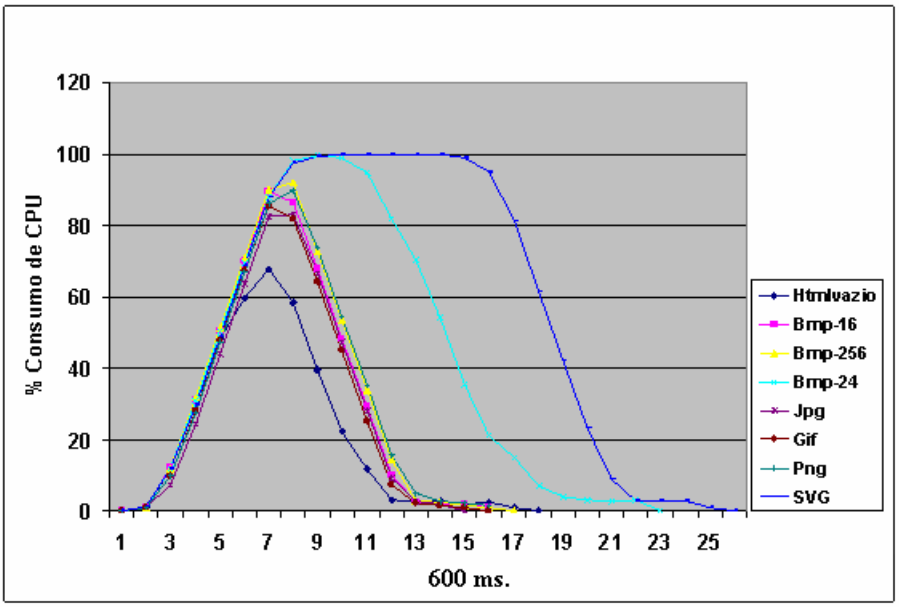

Figura 27 - Ocupação da CPU com o Navegador Firefox 1.5 – Círculos.

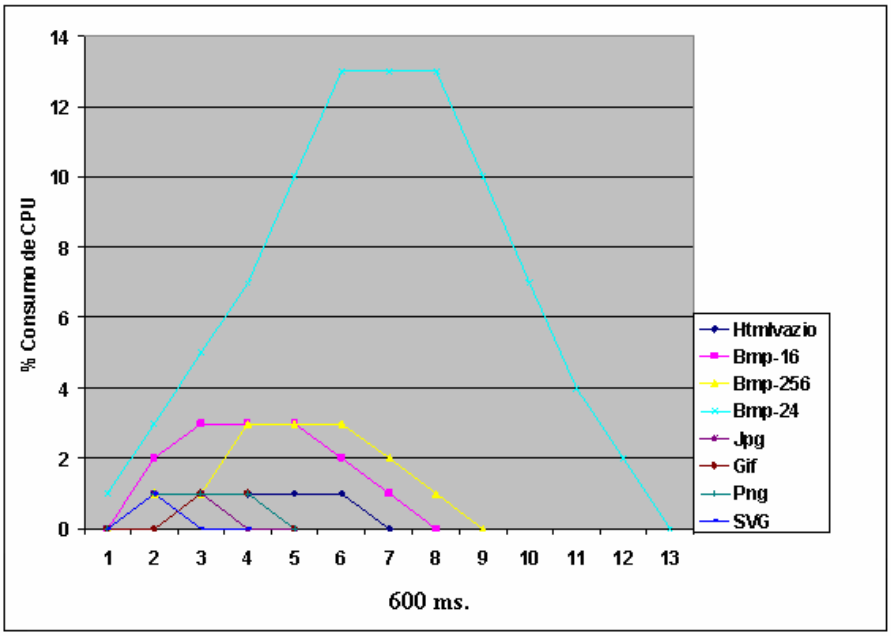

Figura 28 - Consumo Lado CPU Servidor – Círculos.

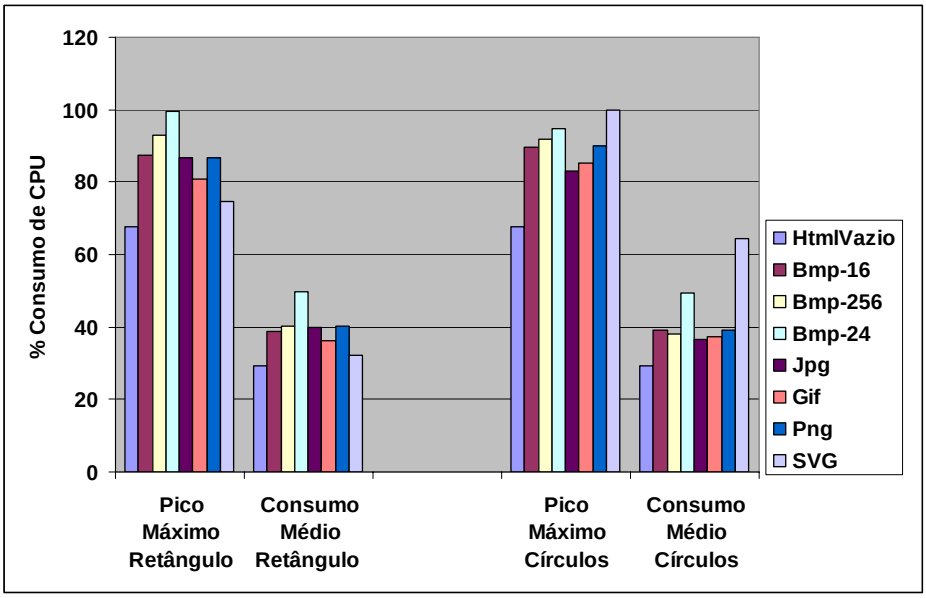

Figura 29 – Pico Máximo e Ocupação média de CPU - Retângulo/Círculos.

Comparando os dois resultados (wimp Retângulo/Círculos) nesse navegador têm-se que:

1. o wimp SVG (Retângulo) obteve o menor consumo médio 32,33% e o menor pico 74,61% de consumo da CPU, valores esses inferiores a todos os formatos analisados. Já o wimp SVG (Círculos) pela sua complexidade obteve o consumo médio de 64,46% e com pico máximo de 100% do uso da CPU (o pico mais alto). O consumo médio praticamente foi o dobro (vide figura 29), enquanto a evolução do tempo de aquisição do SVG com 1 elemento (0,53 segundos) para 19 elementos (1,68 segundos) vide figura 30;
2. na ocupação da CPU no Lado Servidor, o wimp SVG Retângulo obteve o menor percentual. Já os formatos Png, Bmp-256 e Bmp-24 ocuparam os maiores percentuais (vide figura 31);
3. em relação ao Gerenciamento de Memória (vide figura 32) o SVG Retângulo obteve o menor pico de ocupação de memória 27984 Kb e o menor indíce de Pico de Uso de arquivo de Página 18680 Kb. Já o SVG Círculos obteve 28556 Kb. de pico de ocupação de memória e 25112 Kb. de Pico de Uso do arquivo de Página.

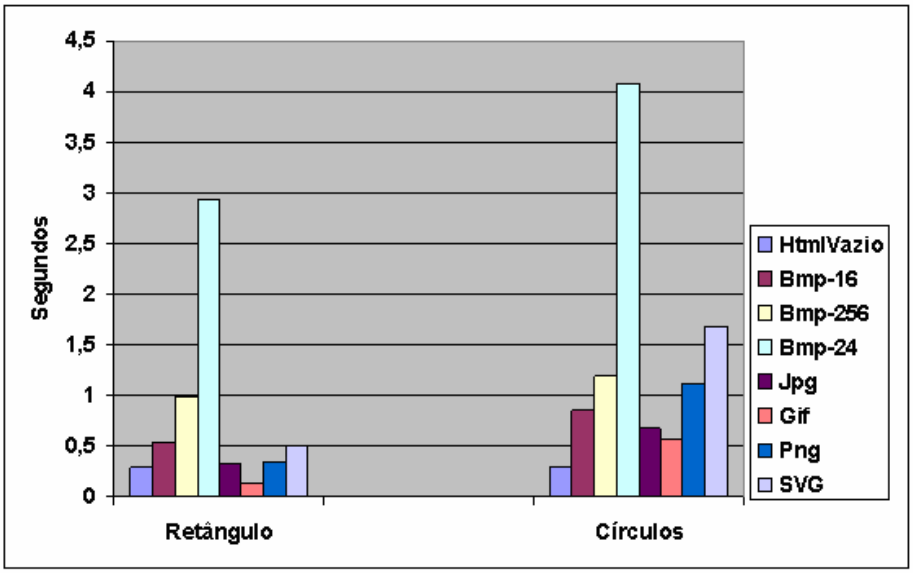

Figura 30 - Tempo de Carregamento das páginas - Retângulo/Circulos.

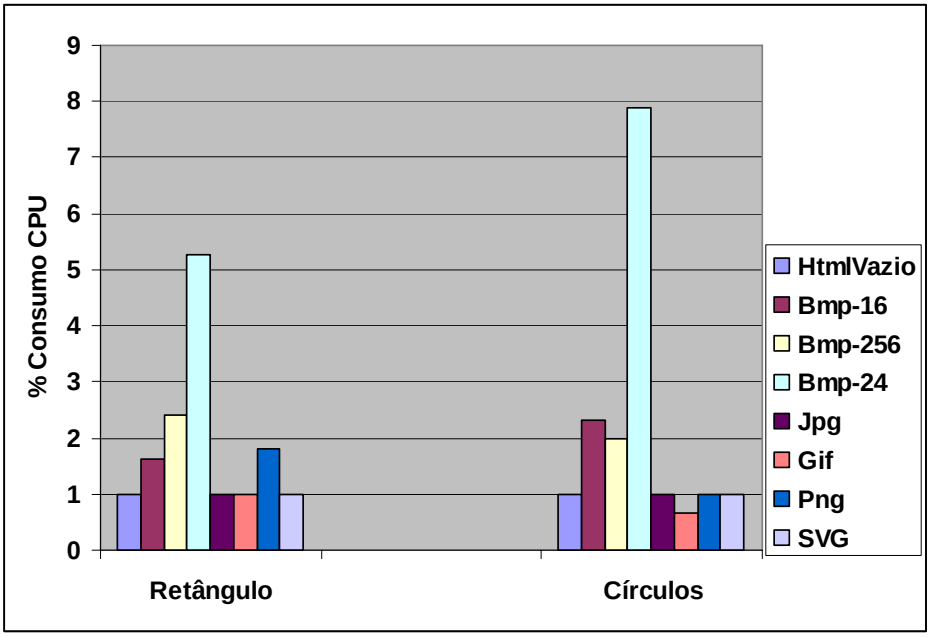

Figura 31 - Consumo CPU Servidor – Retângulo/Círculos.

4.4.2.2 Gerenciamento de Memória

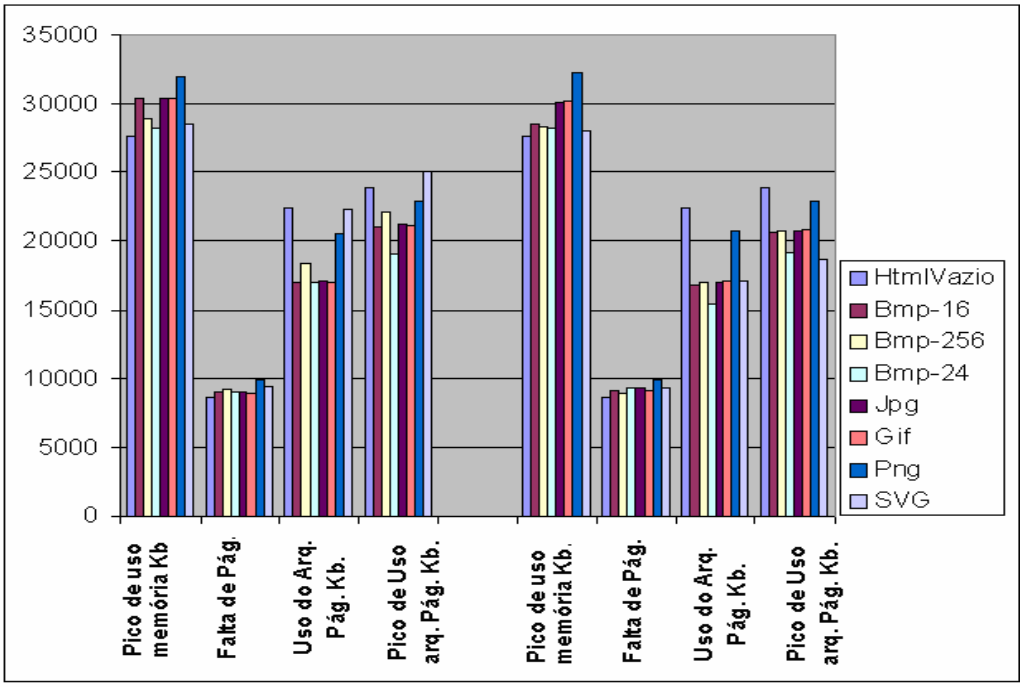

Figura 32 - Gerenciamento de Memória – Cliente Firefox 1.5 - Retângulo/Círculos.

Conforme visualizado na figura 32 existe uma relação na ocupação da memória com os arquivos de cada navegador, no processo de visualização (tamanho dos arquivos e dependência de DLL). Fato observado também com o navegador Internet Explorer.

Na tabela 8 compara-se os dados estatísticos do SVG de 1 e de 19 primitivas gráficas nesse navegador. Conclui-se que:

1. com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,92572 para o consumo médio de 32,33% e um nível de confiabilidade de 97,14% para o wimp Retângulo;
2. em relação ao tempo médio de aquisição de 0,536 seg. (wimp Retângulo), com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,03416 segundos e um nível de confiabilidade de 93,62%;
3. com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 1,25136 para o consumo médio de 64,46% e um nível de confiabilidade de 98,05% para o wimp Círculos;
4. em relação ao tempo médio de aquisição de 1,68 seg. (wimp Círculos), com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,08838 segundos e um nível de confiabilidade de 94,73%.

Tabela 8 - Desempenho do Wimp SVG - Firefox.

Estatísticas	wimp Retângulo	wimp Círculos
Média CPU	32,33%	64,46%
Desvio Padrão	2,587	3,497
Variância	6,693	12,232
Intervalo de Confiança α = 0,05	± 0,92572	± 1,25136
Nível de confiança	97,14%	98,05%
Tempo Médio	0,536	1,68
Desvio Padrão	0,09548	0,247
Variância	0,0121	0,061
Intervalo de Confiança α = 0,05	± 0,03416	± 0,08838
Nível de confiança	93,62%	94,73%

Excluído: *8*
Inserido: *8*
Excluído: *7*
Inserido: *7*
Excluído: *8*

4.4.3 Usando o Navegador Opera

4.4.3.1 Consumo de CPU

Na próxima seqüência de testes utilizando o navegador Opera 9.02 (vide tabelas 15 e 16 do apêndice B) visualizados nas figuras 33, 34, 35 e 36.

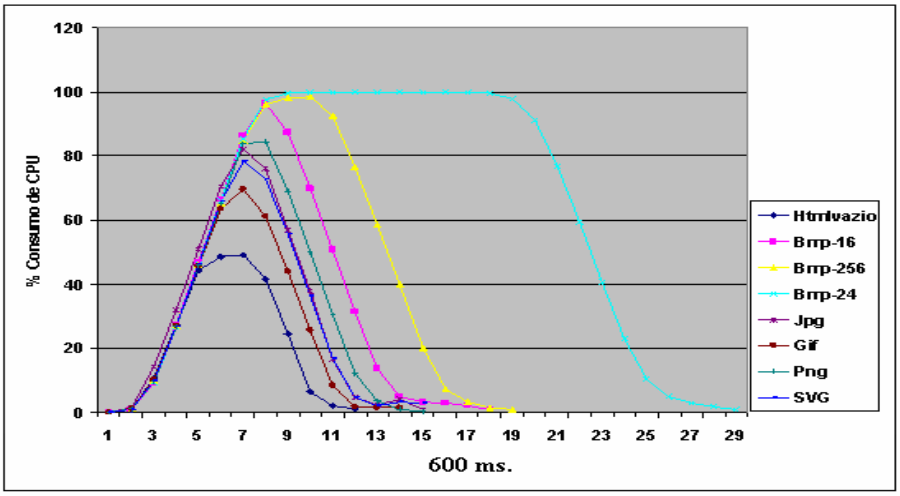

Figura 33 – Ocupação da CPU com o Navegador Opera 9.02 – Retângulo.

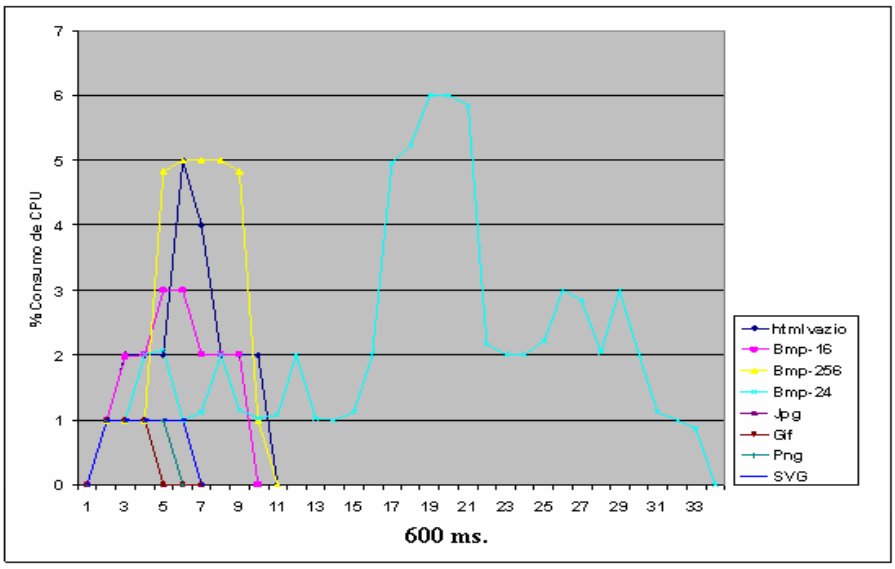

Figura 34 - Consumo CPU Lado Servidor com Cliente Opera 9.02 - Retângulo.

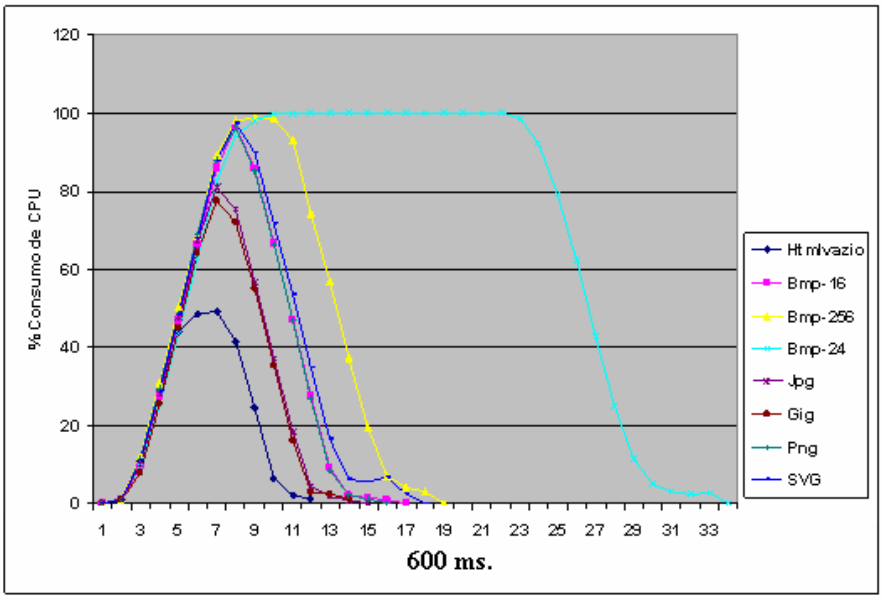

Figura 35 - Ocupação da CPU com o Navegador Opera 9.02 – Círculos.

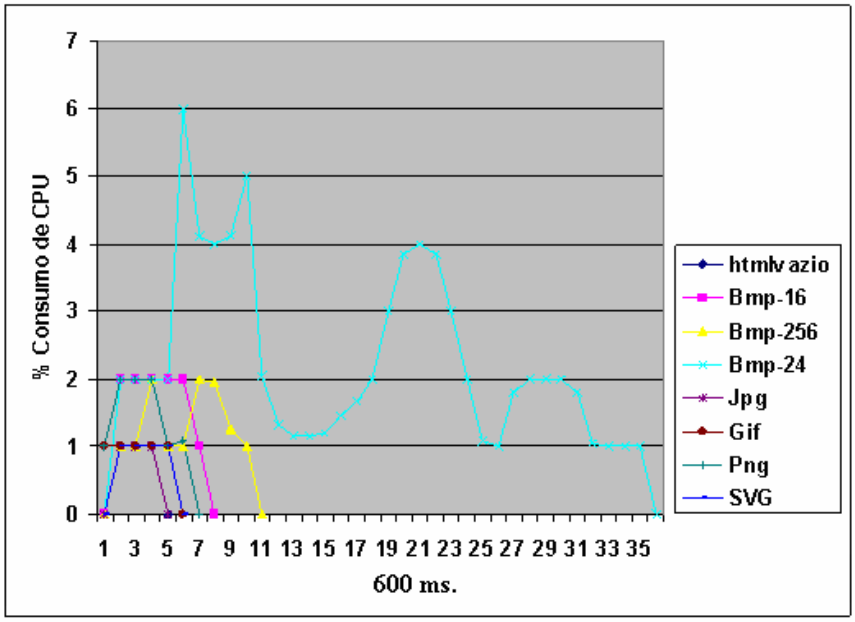

Figura 36 - Consumo CPU Lado Servidor com Cliente Opera 9.02 – Círculos.

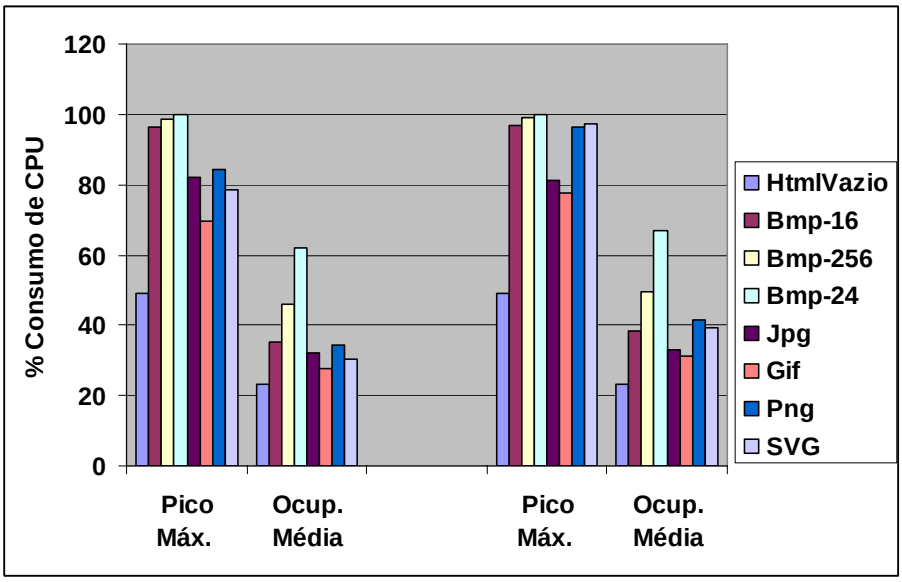

Figura 37 – Pico Máximo e Ocupação média de CPU - Retângulo/Círculos.

Comparando os dois resultados (wimp Retângulo/Círculos) nesse navegador têm-se que:

1. o formato SVG Retângulo, nesse série obteve o pico máximo 78,31%, o consumo médio de 30,24% de ocupação da CPU o melhor indíce dos formatos que vizualizam em 24/32 bits, com o tempo médio de aquisição de 1,05 segundos. Já o formato SVG Círculos obteve 97,45% de pico máximo de utilização, 39,32% de Consumo médio da CPU respectivamente.
2. os formatos JPG, Gif e Png, obtiveram 82%, 69,7% e 84,36% de pico máximo e 32,2% , 27,78%, 34,55 de consumo médio da CPU (vide figura 33). Outro ponto de destaque foi o formato Bmp-24 que teve pico máximo de 100% e 62,21% de consumo médio da CPU por um período de tempo maior 12,07;
3. no lado servidor (ver figura 34), a menor taxa de ocupação do processador foi obtida com o formato Gif (o formato gif não visualiza 24bits) seguido do SVG. Foi também observado que nesse navegador o formato Bmp-24 reteve por mais tempo o processador do servidor. O formato Png nesse navegador obteve um péssimo desempenho. Um fenômeno observado na formação da imagem (Bmp-24) nesse navegador é que a imagem é renderizada de baixo para cima em fatias;
4. no gerenciamento de memória (vide figura 40) observa-se que os formatos

Retângulos GIF e PNG obtiveram os maiores indíces de Ocupação da Memória, já o SVG obteve os menores indices. Comparando o formato Círculos o SVG obteve um melhor desempenho do que o formato PNG. A diferença em termos de pico de uso da memória do SVG (vide tabelas 14 e 15 do apêndice B) (Retângulo 19.464Kb/Círculos 23.148Kb.) foi de apenas 3.684K.

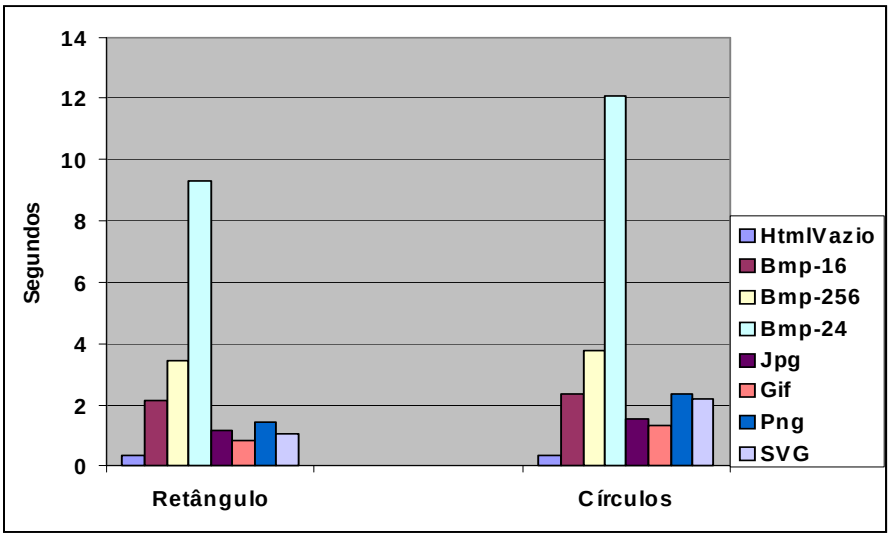

Figura 38 - Tempo de Carregamento das páginas - Retângulo/Círculos.

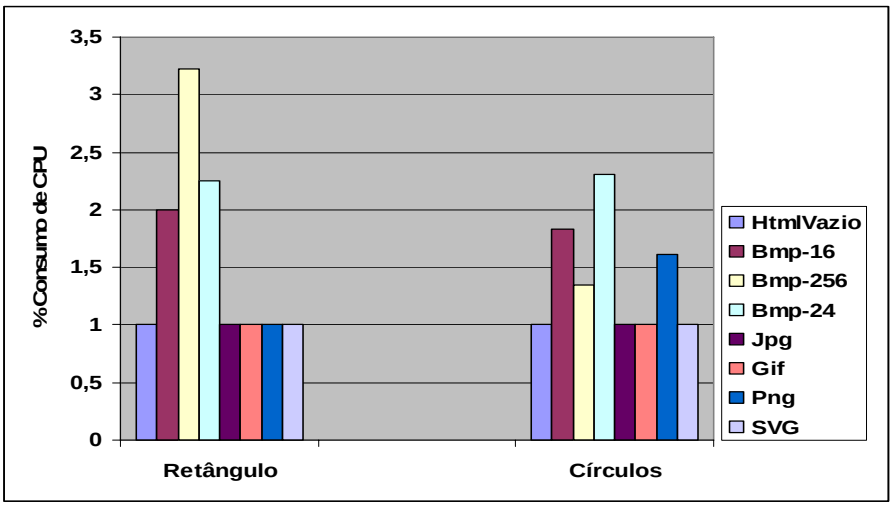

Figura 39 - Consumo CPU Servidor – Retângulo/Círculos.

4.4.3.2 Gerenciamento de Memória

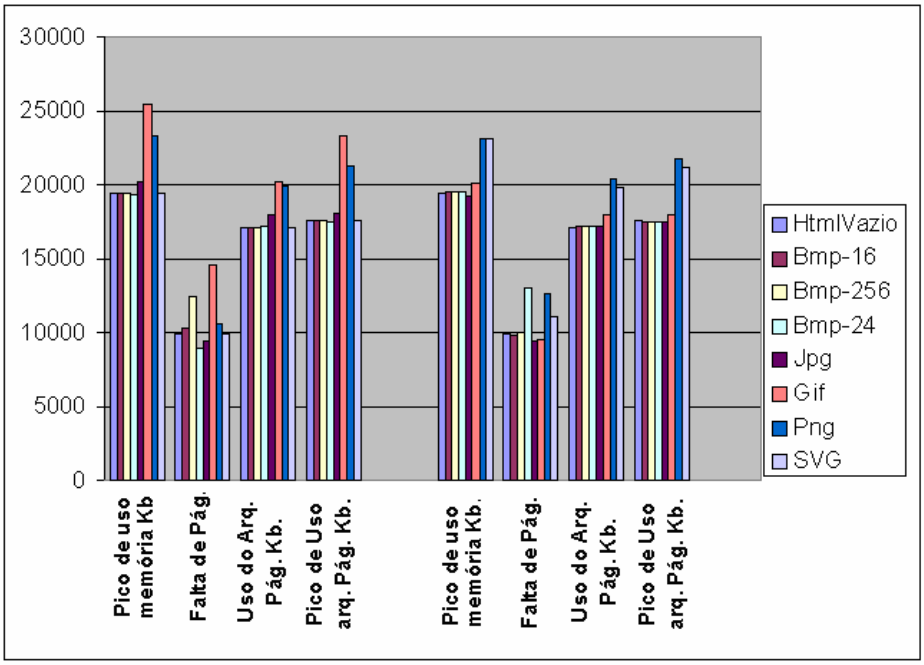

Figura 40 - Gerenciamento de Memória – Opera 9.02 - Retângulo/Círculos.

Na tabela 9 compara-se os dados estatísticos do SVG de 1 e de 19 primitivas gráficas nesse navegador. Conclui-se que:

1. com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 0,95865 para o consumo médio de 30,24% e um nível de confiabilidade de 96,82% para o wimp Retângulo;
2. em relação ao tempo médio de aquisição de 1,057 seg. (wimp Retângulo), com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 1,0337 seg. e um nível de confiabilidade de 97,79%;
3. com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 1,37875 para o consumo médio de 39,32% e um nível de confiabilidade de 96,49% para o wimp Círculos;
4. em relação ao tempo médio de aquisição de 2,198 seg., (wimp Círculos), com um alfa de 0,05 obtêm-se um intervalo de confiança de ± 2,08778 seg. e um nível de confiabilidade de 94,73%.

Excluído: 9
Inserido: 9
Excluído: 8
Inserido: 8
Excluído: 9

Tabela 9 - Desempenho do Wimp SVG – Opera.

Estatísticas	wimp Retângulo	wimp Círculos
Média CPU	30,24	39,32
Desvio Padrão	2,679	3,853
Variância	7,182	14,848
Intervalo de Confiança α = 0,05	± 0,95865	± 1,37875
Nível de confiança	96,82%	96,49%
Tempo Médio	1,057	2,198
Desvio Padrão	0,065	0,308
Variância	0,004	0,095
Intervalo de Confiança α = 0,05	± 1,03374	± 2,08778
Nível de confiança	97,79%	94,98%

4.5 Comparação Final

Na seqüência indicadas pelas figuras 41, 42 e 43 se observa que há dois blocos de dados para cada navegador onde se realiza as comparações em termos de: consumo de CPU lado Cliente, tempo de aquisição e consumo de CPU lado Servidor com os três navegadores. Os resultados mostram que:

1. analisando os dados dos consumos de CPU (Pico máximo, e consumo médio) dos três navegadores com os formatos analisados. Conclui-se que: aplicações que vierem a utilizar o formato SVG no navegador Opera obterão um melhor desempenho arquitetural em sistemas clientes/servidor (vide figura 41);
2. observa-se nos resultados dos testes que o navegador Firefox é o mais rápido na aquisição das imagens (vide figura 42), porém com o maior custo de processamento o que pode afetar máquina cliente com pouco poder computacional;

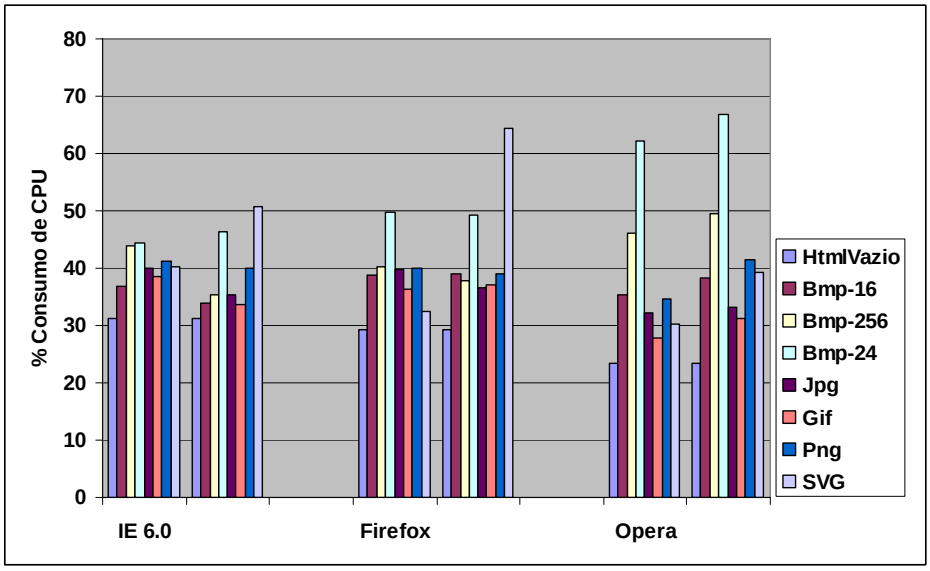

Figura 41 - Comparações analisadas entre os navegadores no impacto da CPU.

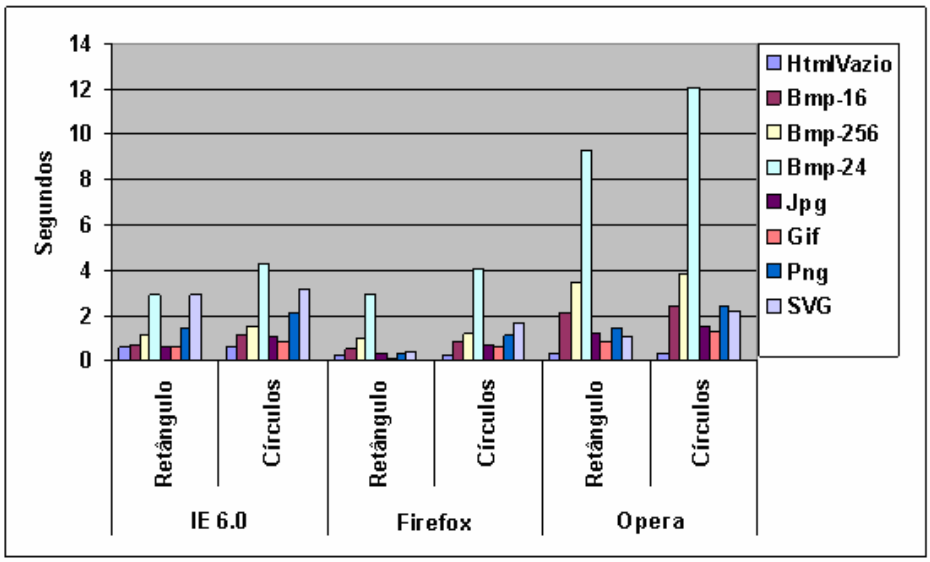

Figura 42 - Comparações de tempo de aquisição de páginas nos navegadores.

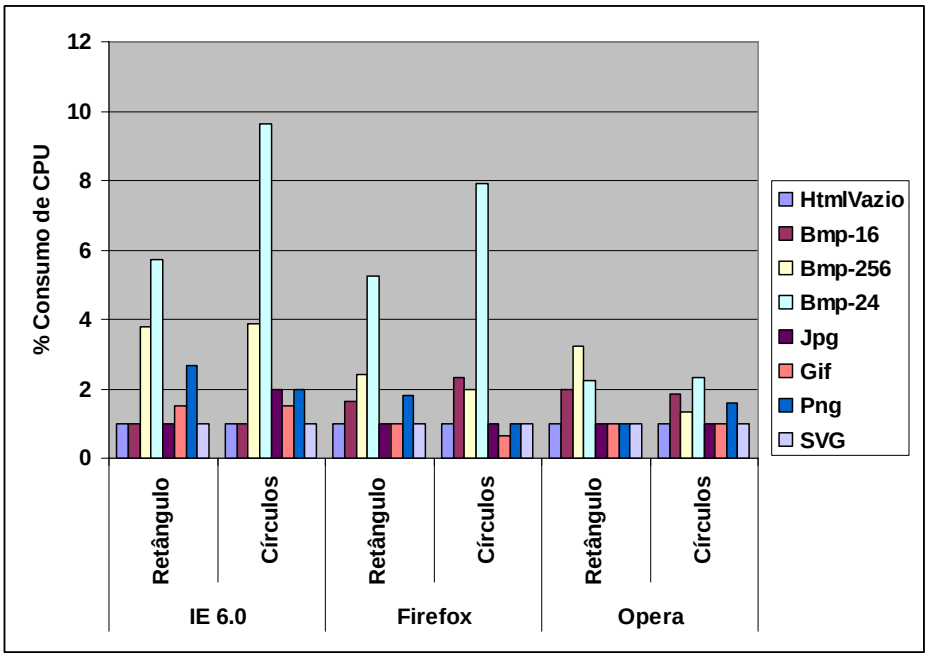

Figura 43 - Comparações de Consumo CPU – lado Servidor nos navegadores.

Um outro teste foi realizado com o objetivo de se obter medidas em um ambiente totalmente distribuído, foi escolhida uma instituição de ensino próxima ao UNIVEM. Essa instituição de ensino possui uma rede com IP: 200.162.174.26 compartilhado (intra-gov). Nos testes (vide tabelas 13 e 14 do apêndice B) foi observado que independente de horários e dos dias da semana a rede tem como característica uma alta latência. Já que essa variável não possui controle, tornou-se uma limitante para estabelecer qualquer comparação, mas algumas conclusões foram extraídas, como por exemplo:

1. em função do aumento dos tempos de respostas na aquisição em cada amostra, foi observado que os picos de consumo diminuíram, porém aumentando a média final de ocupação da CPU;
2. o servidor ficou ocupado o tempo respectivo das requisições do cliente;
3. quando a latência estava excessivamente alta, a renderização do SVG (Círculos) é formado pela seqüência de primitivas serializadas que chegam à máquina cliente à medida que são transmitidas pelo servidor.

As figuras 44 e 45 representam os melhores momentos nesse ambiente de teste. Esse cenário de testes foi o mais crítico de todos.

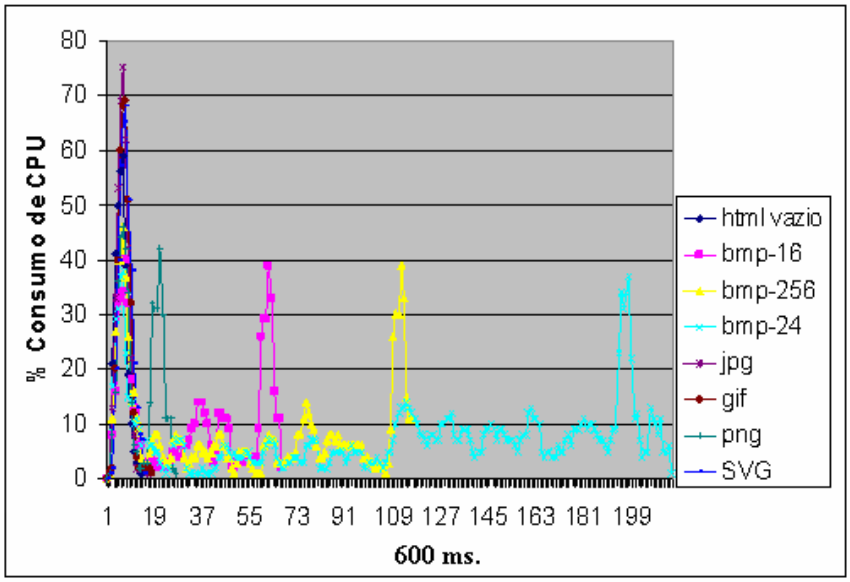

Figura 44 - Ocupação da CPU com o Navegador Opera 9.02 – Retângulo.

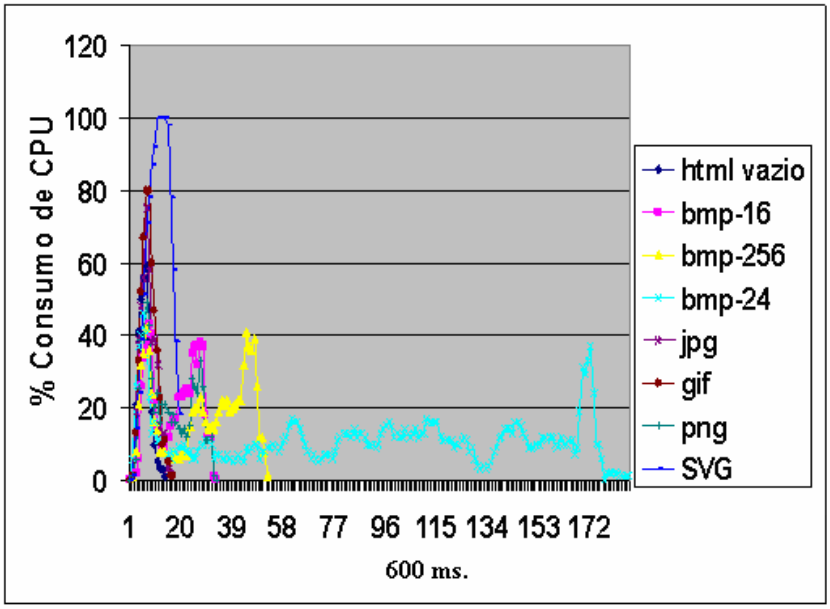

Figura 45 - Ocupação da CPU com o Navegador Opera 9.02 – Círculos.

5. Conclusões e Trabalhos Futuros

A utilização de gráficos na *Web* tem diferentes usos: visualizar dados (mapas, diagramas, ilustrações, imagens, animações, interações, etc), também podem ser utilizados para criar ambientes e mundos virtuais em games, simuladores e monitores de estado. No caso de simuladores e monitores que utilizam os formatos *raster*, o processamento se efetua no servidor, a cada alteração da interação o cliente apenas faz o *download* das atualizações, isso não ocorre com aplicações que utilizam o SVG que apenas solicitam dados semânticos de atualizações. Essa abordagem de interação com imagens "rasterizadas" é pesada e pouca eficiente.

Em toda a literatura pesquisada (QIU, 2005), (GARCIA, 2006), (MARINHO et al., 2006), nesta dissertação, somente o relatório de Tazkiya (ALI, 2005), enfatizou o alto consumo de CPU no lado cliente com o uso do SVG, porém, sem esclarecer maiores detalhes. Já nesse trabalho, além da coleta dados e consolidação dos mesmos, conclui-se também que além do consumo de processamento exigido por esse formato no lado cliente, o mesmo pode ter desempenho diferente dependendo do navegador a ser utilizado o que afeta a usabilidade principalmente nos dispositivos portáteis digitais que possuem pouco poder de processamento. Destaca-se que um alto consumo de CPU aumenta o consumo da bateria, o que diminui a usabilidades desses dispositivos.

Os resultados também indicam que para todos os tamanhos de arquivos analisados, o fator predominante é o tempo gasto nas transações para enviar o conteúdo do servidor até o cliente (latência). Ressalta-se que em todos os testes realizados, o servidor *Web* consumiu taxas inferiores a 20% de carga do processador no tempo total de resposta (Bmp-24). Já o SVG consumiu em média 1%.

Aplicando a metodologia definida nos procedimentos dos testes o tempo médio gasto na realização de cada amostra foi de 20 minutos e o total de amostras foi de 1920.

Os aplicativos Internet Explorer, Firefox e Opera possuem respectivamente, entre arquivos executáveis e dependências de DLL 38.040.897 Mb, 29.187.866 Mb, 18.886.656 Mb.

Uma particularidade encontrada nos navegadores Opera/Firefox é que ambos fazem uso das DLL (IMM32.DLL, wsock32.dll, ADVAPI32.dll, GDI32.dll, comctl32.dll, comdlg32.dll, kernel32.dll, msvcrt.dll, ntdll.dll, ole32.dll, RPCRT4.dll, SHELL32.dll,

SHLWAPI.dll, USER32.dll, VERSION.dll, WINMM.dll, WS2_32.dll e WS2HELP.dll) pertencentes ao navegador Internet Explorer.

A tabela 10 mostra algumas observações interessantes. Vários aspectos dos testes são comparados em termos de desempenhos dos conteúdos analisados.

Tabela 10 - Análise final dos Wimp Retângulo/Círculos

I.E 6.0	**% Pico Máx. CPU**	**% Consumo Médio CPU**	**Tempo Médio Aquisição - Seg.**	**% Ocupação CPU Servidor**	**Pico Ocupação Memória**
Htmlvazio	61,46	31,16	0,57	1	19.464K
SVG Ret.	70,29	40,36	2,94	1	24.852K
SVG Circ.	92,13	50,81	3,23	1	25.648K
Firefox 1.5	**% Pico Máx. CPU**	**% Consumo Médio CPU**	**Tempo Médio Aquisição - Seg.**	**% Ocupação CPU Servidor**	**Pico Ocupação Memória**
Htmlvazio	67,71	29,18	0,29	1	27.628K
SVG Ret.	74,61	32,33	0,53	1	27.984K
SVG Circ.	100	64,46	1,68.	1	28.556K
Opera 9.02	**% Pico Máx. CPU**	**% Consumo Médio CPU**	**Tempo Médio Aquisição - Seg.**	**% Ocupação CPU Servidor**	**Pico Ocupação Memória**
Htmlvazio	49,27	23,37	0,33	1	19.464K
SVG Ret.	78,31	30,24	1,05	1	19.464K
SVG Circ.	97,45	39,32	2,19	1	23.148K

Excluído: *10*
Inserido: *10*
Excluído: *9*
Inserido: *9*
Excluído: *10*
Excluído: I.E 6.0 ... [21]
Tabela formatada
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt
Formatado: Fonte: 11 pt

- conclui-se que o navegador Firefox obteve os melhores tempos de aquisição do formatos gráficos wimp (Retângulo/Círculos), porém com um custo elevado de processamento, sendo que seu *parser* primeiro faz a aquisição do arquivo para depois renderiza-lo, opostamente aos navegadores IE 6.0/Opera (vide figuras 16,19,25,27,33 e 35) que renderizam as imagens de acordo com a chegada dos pacotes do servidor (qtde. de loop com IDL) processo de recuperação da imagem (codificar/decodificar/renderizar);
- observa-se que nos três navegadores, o menor índice obtidos na relação entre 1 e 19 elementos vetoriais (Retângulo/Círculos), a menor taxa de consumo de CPU/Memória foi obtido pelo navegador Opera;
- Um outro fato significante obtido nos testes de aquisição do arquivo html vazio nos três navegadores têm-se que o Opera obteve o melhor desempenho:

Devido à inexistência de espaço vazio das imagens Retângulos *(raster)* na renderização, observou-se que esses formatos geraram uma carga maior em relação aos formatos Círculos do tipo *raster*, evidenciando que o consumo de CPU de cada formato está relacionado com o compilador de cada navegador, o tamanho do arquivo, o algoritmo de compressão, a complexidade da imagem, a sua extensão e a latência da rede. Salienta-se que o formato SVG não possui algoritmo de compressão, sendo a sua renderização mais pesada em relação a alguns formatos *raster*.

Excluído: ¶
Formatado: Fonte: Itálico
Formatado: Fonte: Itálico

O SVG por possuir um arquivo de menor tamanho, o torna viável para o armazenamento em disco e desenvolvimento de componentes gráficos de interação do tipo WIMP em aplicações *Web*, pois os testes revelaram a pouca ocupação na banda de transmissão.

No geral, foi observado durante os testes que o desempenho arquitetural utilizando o formato SVG dependerá fundamentalmente do número de primitivas gráficas contidas na imagem armazenada.

Uma característica notável com o formato SVG que independente do tamanho da imagem, o tamanho do arquivo é o mesmo, opostamente aos formatos *raster*.

Foi observado nos estudos que apesar do custo de processamento do SVG inicial ser mais alto em relação aos formatos *raster* alguns testes, no lado cliente, e que dependendo do desígnio da aplicação o SVG é uma boa solução e esse custo inicial não será problema. Geralmente qualquer interação com formatos *raster* usará no mínimo dois estado com duas imagens, o que consumiria o dobro do tempo de aquisição e o dobro de consumo de CPU;

Uma outra característica do formato SVG é que mesmo pode ser publicado em qualquer servidor *Web* sem a necessidade de um servidor de imagens.

As imagens formadas pelo SVG podem ser impressas em alta qualidade sem distorção e em qualquer resolução nas mesmas medidas em que são apresentadas na tela. Demonstrando o estilo de interação WYSIWYG - *What You See Is What You Get* (O que Voce Vê é o Que Voce Pega);

SVG possui uma série de vantagens em relação aos formatos gráficos mais usados na *Web* como PNG, JPEG ou GIF. Por ser um formato de texto simples, os arquivos do SVG são legíveis a edição humana e o seu conteúdo pode ser localizado por motores de procura;

O formato SVG possui recurso de *zoom in zoom out* e são escalonáveis, ou seja, os

usuários podem aproximar uma área específica em particular de uma imagem, como um mapa, e não ter nenhuma distorção. Por exemplo. Como limitação do uso do SVG destaca-se que, por ser um arquivo formatado em texto puro não oferece segurança o mesmo pode ser facilmente modificado.

Finalizando, com a aplicação deste trabalho em aplicações *Web*, espera-se uma possível redução no custo computacional dos servidores, clientes e uma melhor otimização da banda, habilitando a transmissão apenas de dados incrementais entre o cliente e o servidor, visualizações mais perfeitas para Interface Gráfica de Usuário e portabilidade para processamento gráfico.

A seguir menciona-se algumas dificuldades que foram encontradas ao longo deste trabalho:

1. a primeira foi de compreender o comportamento das imagens, principalmente a do formato SVG, sendo necessário estudar todas as formas de imagens geradas pelo Inkscape e qual o impacto que elas representam em um determinado tipo de *hardware*;
2. impossibilidade de medir exclusivamente as renderizações das imagens, devido ao não acesso ao código fonte das DLL responsáveis por essa *thread*;
3. problemas com os *parser* dos navegadores: Foram realizados 206 testes em cada navegador com o padrão recomendado na *Full 1.2 Specification W3C Working Draft* de 13 abril de 2005 (SVG, 2005) e no *Test Suite* 13 de dezembro de 2006 (SVG, 2006), os resultados revelaram falhas em alguns testes devido a rigidez do *parser* do Internet Explorer 6.0/Firefox na animação/interação do SVG;
4. a realização de testes em um ambiente distribuído;
5. pouca literatura sobre o SVG;
6. a não resposta de vários e-mail para diversos autores que publicaram trabalhos sobre o estado da arte do SVG.

Formatado: Fonte: Itálico

Espera-se para trabalhos futuros, a partir desse estudo e testes ser complementado e aprofundado. Pode-se citar as seguintes sugestões:

1. aplicar esse conhecimento adquirido em testes com sistemas derivados do Unix;
2. realizar medições com processadores Dual Core;

3. implementar, analisar o comportamento em PDA e celulares;
4. comparar o SVG com as técnicas de Graphics Pipeline Interception;
5. implementação de um processador semântico (SVG) em FPGA.

REFERÊNCIAS BIBLIOGRÁFICAS

ALI, Tazkiya Sheik Muhammed: **A SVG web environment for Zformal specification Language**, 7th International Conference on Formal Engineering Methods (ICFEM'05), LNCS, Springer-Verlag, pages 480-494, Manchester, United Kingdom, November 2005.

APPLE Computer, Inc. **Macintosh Human Interface Guidelines**, Inside Macintosh: Overview, Inside Macintosh: Toolbox Essentials, Programmer's Introduction to the Macintosh Family. Addison-Wesley. 1992.

APPLETS. **The Java Tutorial - Deployment – Applets**. Disponível em: <http://java.sun.com/docs/books/tutorial/deployment/applet/index.html>. Acesso em: 10/09/2006.

BARDINI, Thierry and Friedewald, Michael: **History of Technology, Chronicle of the Death of a Laboratory: Douglas Engelbart and the Failure of the Knowledge Workshop**, Volume Twenty-three, 2002.

BATIK, **Batik overview**. Disponível em: <http://xml.apache.org/batik/index.html>. Acesso em: 01/08/2005.

BEANS, Beth Stearms, **Tutorials & Code Camps – JavaBeans 101**, october 2000, Disponível em:<http://java.sun.com/developer/onlineTraining/Beans/bean01/page2.html#def>

. Páginas: 01,02,03 e 04. Acesso em 20/04/2006.

BORUSCH, Daniel, Lung, Lau Cheuk, Bessani, Alysson Neves, Fraga, Joni da Silva: **Integração das Especificações ROMIOP e ETF para Difusão Atômica no CORBA,** 2005

BUSH, V. **As We May Think by**, Vannevar Bush. *The Atlantic on-line Journal (The Atlantic Monthly July 1945*. Disponível em:< http://www.theatlantic.com/doc/194507/bush>.

Acesso em: 10/10/2005.

BOSHERNITSAN, Marat. Downes Michael S, **Visual Programming Languages: A Survey** Report No. UCB/CSD-04-1368,Computer Science Division, EECS. University of California, Berkeley, CA 94720-1776 USA, 2004.

Formatado: Cor da fonte:

Formatado: Cor da fonte:
Formatado: Sem sublinhado
Formatado: Português

Excluído: e
Código de campo alterado
Formatado: Português
Formatado: Sem sublinhado, Português (Brasil)
Formatado: Português
Excluído: Acessado
Excluído:

Formatado: Cor da fonte:
Excluído: Acesso

BLACKWELL, Alan Frank: **Metaphor in Diagrams**, Dissertation - Doctor of Philosophy, Darwin College. Cambridge University of Cambridge, September 1998. Disponível em:< http://citeseer.ist.psu.edu/cache/papers/cs/3672/http:zSzzSzwww.mrc-cbu.cam.ac.ukzSzprojectszSztwdzSzthesiszSzblackwell-thesis.pdf/blackwell98metaphor.pdf>.Acesso em: 22/03/2006.

COULOURIS, Geoge, Dollimore; Kindberg, Tim: **Distributed Systems Concepts and Design,** segunda edição, Addison-Wesley, 1998.

CHANG, S. K. **Principle of Visual Programming Systems,** Prentice Hall International, Inc, New York. 1990.

CHONG, Stephen and Purcell, Ricardo: **A Framework for Creating Natural Language User Interfaces for Action-Based Applications**. 2004. Disponível em:<http://citeseer.ist.psu.edu/cache/papers/cs/32852/http:zSzzSzwww5.cs.cornell.eduzSz~schongzSzpubszSzamilp03.pdf/a-framework-for-creating.pdf>. Acesso em: 12/02/2006.

CLI. Landley, Rob W. **The Art of Unix Usability.** April 18 2004. **Command-Line Interfaces** Disponível em: <http://www.catb.org/~esr/writings/taouu/html/>. Acesso em: 09/2006.

COM, **COM: Component Object Model Technologies.** Disponível em: <http://www.microsoft.com/com/default.mspx>. Acesso em: 10/01/2006.

CONVERSY, Stéphane and Fekete, Jean-Daniel: **The svgl toolkit: enabling fast rendering of rich 2D graphics.** 2001. Disponível em:< http://www.lri.fr/~conversy/publications/emn2002.pdf>. Acesso em:20/02/2006.

DEITEL, H.M: **Java: como programar**, 6º. ed. São Paulo, Pearson Prentice Hall, 2005.

DEEPFREEZE, **Deep Freeze Professional 2000XP**. Disponível em: < http://www.software.ufl.edu/deepfreeze/>. Acesso em: 12/06/2006.

DUMAS, Arthur: **Programando Winsock**, tradução Jorge Cockles Costa de Oliveira, Rio de Janeiro, Axcel Books, 1995.

Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Português
Excluído:

DRAGICEVIC, P. and Fekete, J.-D., **Input Device Selection and Interaction Configuration with ICon**, Proceeding of IHM-HCI, 2001. Disponível em:< http://citeseer.ist.psu.edu/cache/papers/cs/24970/http:zSzzSzwww.emn.frzSzfeketezSzpszSzminput-tr.pdf/dragicevic01input.pdf>. Acesso em: 20/03/2006.

DRAGICEVIC, Pierre**: Un modèle d'interaction en entrée pour des systèmes interactifs Mult. -dispositifs hautement configurables**, Thèse de Doctorad de l'Université de Nantes. 2004. Disponível em:< http://www.dgp.toronto.edu/~dragice/these/html/memoire_dragicevic.html >. Acesso em 12/09/2005.

EMMUS**, Introduction to user centred design, ISO 13 407 for Multimedia,** Disponível em: <http://www.ucc.ie/hfrg/emmus/methods/intro.html>. Acesso em: 08/10/2005.

GARCIA, Rodrigo, **SVG for SCADA Applications A practical approach.** Ph.D thesis Disponível em**:** <http://www.svgopen.org/2004/papers/SVGforSCADA/#substyle>. Acesso em: 12/06/2006.

GUITIMELINE, Disponível em: <http://toastytech.com/guis/guitimeline.html>. Acesso em: 10/11/2005

GULLIKSEN, Jan, **Key Principles for User-Centred Systems Design**, Jan Gulliksen, Bengt Göransson, Inger Boivie, Stefan Blomkvist, Jenny Persson & Åsa Cajander, To be published in BIT ©The Authors, 2003, material may be cited but not altered – version A. Disponível em:<http://www.it.uu.se/research/hci/acsd/englishsummary.html>.Acesso em: 13/07/2005.

HISTORY, **W3C Scalable Vector Graphics (SVG) – History**. Disponível em:< http://www.w3.org/Graphics/SVG/History>. Acesso em: 02/03/2005.

HOPSON, Hopson, K. C e Ingram, Stephen E.: **Desenvolvendo Applets com Java.** Editora Campus, 1997.

HORN, Robert E, **Think Link, Invent, Implement, and Collaborate! Think Open! Think Change! Think Big!**. Disponível em;< http://www.stanford.edu/~rhorn/a/recent/spchThinkEngelbart.pdf>. Acesso em: 10/06/ 2006.

IDL. Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/idl/>. Acesso em: 05/04/2006.

Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Francês (França)
Formatado: Francês (França)
Formatado: Português
Formatado: Cor da fonte: Azul, Português (Brasil)
Excluído: Acessado
Formatado: Português
Formatado: Português
Formatado: Cor da fonte: Azul, Português (Brasil)
Formatado: Português
Excluído: Acesso
Formatado: Cor da fonte:
Excluído: Acesso
Excluído: Acesso
Formatado: Cor da fonte:
Formatado: Português
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Português
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Português

INTEL, **Using the RDTSC Instruction for Performance Monitoring.** Copyright (c) Intel Corporation 1997. Disponível em: < http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf>. Acesso em: 20/06/2006.

INTEL**, Intel Architecture Software Developer's Manual Volume 3: System Programming Intel® Architecture Optimization** *Reference Manual, 1999,* Copyright © 1998, 1999 Intel Corporation All Rights Reserved Issued in U.S.A. Order Number: 245127-001. Disponível em:<ftp://download.intel.com/design/PentiumII/manuals/>. Acesso em: 10/09/2006.

INTEL, Intel Architecture Software Developer's Manual Volume 1: Basic Architecture. Volume 1: Basic Architecture. 1999. Disponível em:<ftp://download.intel.com/design/PentiumII/manuals/>. Acesso em: 10/09/2006.

INTEL**, IA-32 Intel® Architecture Software Developer's Manual Volume 2: Instruction Set Reference 2002.** Copyright © Intel Corporation, All Rights Reserved Issued in U.S.A.Order Number: 245127-001. Disponível em:<ftp://download.intel.com/design/PentiumII/manuals/>. Acesso em: 10/09/2006.

JANSEN, Bernard, J.: **The Graphical User Interface: An Introduction,** SIGCHI Bulletin. 30(2), 22-26. 1998.

JAKOBSON, R. **Lingüística e Comunicação**. Cultrix, São Paulo. 1970.

LÉVY, Pierre. **A Máquina Universo: Criação, Cognição e Cultura informática**. Porto Alegre, Ed. Artmed, primeira edição, 1998.

LICKLIDER, J.C.R., "**Man-Computer Symbiosis**", IRE Transactions on Human Factors in Electronics, HFE-1 (March 1960): 4-11. Reprinted in In Memoriam: J.C.R. Licklider: 1915-1990. 1-19. Palo Alto, Digital Systems Research Center, 1990.

MENASCÉ, Daniel A.: **Planejamento de capacidade para serviços na Web: métrica, modelos e métodos,** Editora Campus, Rio de Janeiro, 2002.

MACHIRAJU, Vijay: **A survey on Research in graphical User Interfaces**. 1996. Disponível em:<http://citeseer.comp.nus.edu.sg/cache/papers/cs/3670/http:zSzzSzwww.cs.

Formatado: Cor da fonte:
Excluído: Acessado
Excluído:
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Sublinhado, Cor da fonte: Azul
Excluído: Acessado
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Português
Formatado: Cor da fonte:

utah.eduzSz~machirajzSzreszSzuizSzui.pdf/machiraju96survey.pdf>. Acesso em: 22/03/2006.

MARINHO, Filipe, Viegas Paulo, Lopes, João Correia. **Utilização de SVG na Visualização de Sinópticos**. Disponível em:< http://vecpar.fe.up.pt/xata2006/papers/15.pdf>. Acesso em: 10/12/2006.

MORAN, T. **"The Command Language Grammars: a representation for the user interface of interactive computer systems**. International Journal of Man-Machine Studies, 1981.

OLIVEIRA, José Ivo Fernandes; e MUCHERONI, Marcos Luiz: – Interface Gráfica Distribuída, I Workshop de Dissertações em Computação, SP.2006.

OMG, **Object Management Group, Object Management Architecture**. Disponível em: <http://www.omg.org/oma/>. Acesso em: 01/02/2006.

PARC. Disponível em: <http://www.parc.xerox.com/about/history/default.html>. Acesso em: 02/06/2006.

PEIRCE, C.S. (1931-1958). **Collected Papers**. Edição brasileira: **Semiótica**. São Paulo, Ed. Perspectiva (coleção estudo, n.46) 1977.

PIÁSIL, Frantisek: **An Architectural View of Distributed Objects and Components in CORBA, Java RMI, and COM/DCOM**, 1999. Disponível em: <http://citeseer.csail.mit.edu/cache/papers/cs/7437>. Acesso em: 10/04/2006.

PREECE, J.; Rogers, Y.; Sharp, E.; Benyon, D.; Holland, S.; Carey, T. **Human-Computer Interaction**. Addison-Wesley. 1994.

PRITCHARD, Jason**. COM and CORBA Side by Side, Architectures, Strategies, and Implementations .** Addison-Wesley, 1999.

QIU, Xiaohong: **Message-based MVC Architecture for Distributed and Desktop Applications**, Thesis Doctoral of Philosophy in Computer Science of Syracuse University, 2005. Disponível em:< http://grids.ucs.indiana.edu/~xqiu/complete_thesis.pdf >. Acesso em: 20/02/2006.

Excluído: Acessado

Excluído: Acessado

Formatado: Cor da fonte:

Formatado: Português

Formatado: Cor da fonte:

Excluído: Acesso

Excluído: Acesso

Formatado: Cor da fonte:

Formatado: Cor da fonte:

Excluído: Acesso

Formatado: Português

Formatado: Inglês (E.U.A.)

Formatado: Cor da fonte:

Excluído: Acessado

SILVEIRA, L. F. B. da: **Mente, Universos e Verdade nas Relações Semióticas**. *Caderno da 4ª Jornada do CEPE*, PUC-SP, 6-13. (2001)

SOCKET. **Lesson**: **All About Sockets,** Java tutorials Custom Networking. Disponível em: <http://java.sun.com/docs/books/tutorial/networking/sockets/index.html>. Acesso em: 06/09/2006.

SUTHERLAND, Ivan Edward. **Sketchpad: A Man-Machine Graphical Communication System**. New York: Garland Publishers, 1980. Disponível em: <www.cc.gatech.edu/classes/cs6751_97_fall/projects/abowd_team/ivan/ivan.html>. Acesso em: 03/10/2005.

SVG, **W3C Recommendation 14 January 2003.** Disponível em:<http://www.w3.org/TR/SVGMobile/>. Acesso 01/10/2006

SVG, **Scalable Vector Graphics (SVG) Full 1.2 Specification W3C Working Draft *13 April 2005*.** Disponível em: <http://www.w3.org/TR/SVG12/>. Acesso em 01/09/2005.

SVG, **Explore the possibilities of SVG SCALABLE VECTOR GRAPHICS**. Disponível em: <http://www.adobe.com/svg/overview.html>. Acesso em: 08/12/2005.

SVG, **W3C Scalable Vector Graphics (SVG) 1.1 Test Suite 13 Dec 2006.** Disponível em: < http://www.w3.org/Graphics/SVG/Test/>. Acesso em: 02/01/2007.

SVGMODELS. Chatty, Stephanie, Lemort, Alexandre, Sire, Stephane, Vinot, Jean-Luc, **Combining SVG and models of interaction to build highly interactive user interfaces**, Disponível em: <http://www.svgopen.org/2005/papers/CombiningSVGModelsBuildInteractiveUserInterfaces/index.html#S2.1>. Acesso em: 10/03/2006.

TANENBAUM**,** Andrew S.: **Sistemas Operacionais Modernos,** prentice-Hall, Inc, 1992 traduzido em língua portuguesa por Nery Machado Filho. 1995.

TANENBAUM**,** Andrew S.: **Sistemas Operacionais Modernos,** prentice-Hall, Inc, 2003, traduzido em língua portuguesa por Ronaldo A. L. Gonçalves, Luiz A. Consularo.

TANENBAUM**,** Andrew S.: **Redes de Computadores,** Ed. Campus, 2003, tradução: Vandenberg D. de Souza. Rio de Janeiro, 2003.

Código de campo alterado
Formatado: Cor da fonte:
Formatado: Cor da fonte:
Excluído: Acesso
Formatado: Cor da fonte:
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Cor da fonte:
Excluído: Acesso
Formatado: Cor da fonte:
Excluído: Acesso
Formatado: Português
Formatado: Inglês (E.U.A.)
Formatado: Cor da fonte:
Excluído: Acessado
Formatado: Cor da fonte:
Excluído: Acesso
Formatado: Inglês (E.U.A.)

TANENBAUM**,** Andrew S. Stem, Maarten Van: **DISTRIBUTED SYSTEMS PRINCIPLES AND PARADIGMS,** Ed. Prentice Hall. 2003.

USABILITY, **Usability Glossary: ISO 13407 Human-Centered Design Process**. Disponível em: <http://www.usabilityfirst.com/glossary/term_1196.txl>. Acesso em: 07/01/2006.

ULLMER, Brygg, and Ishii Hiroshi, **Emerging Frameworks for Tangible User Interfaces**, **In "Human-Computer Interaction in the New Millenium**," John M. Carroll, ed.; ©Addison-Wesley, August 2001, pp. 579-601. Portions reprinted with permission from IBM Systems Journal, Vol. 39, No. 3/4, 2001.

WEBSTER, Timothy. **Guide to Graphics PNG, GIF & JPEG**. Hayden Books, 1997.

WINOGRAD, T**. The Stanford HCI Design Learning Space: Metaphor**. Disponível em: <http://hci.stanford.edu>. Acesso em: 18/11/2005.

Formatado: Cor da fonte:

Formatado: Cor da fonte:

Apêndice A – Código Aplicação Java

HelloServer.java

```
// Copyright and License Sun Microsystems.
// HelloServer.java
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;
class HelloImpl extends HelloPOA {
  private ORB orb;
  public void setORB(ORB orb_val) {
    orb = orb_val;
  }
   // implement sayHello() method
  public String sayHello() {
    return "\nHello world !!\n";
  }
     // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
}
public class HelloServer {
  public static void main(String args[]) {
    try{ // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get reference to rootpoa & activate the POAManager
      POA
rootpoa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
      // create servant and register it with the ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);
      // get object reference from the servant
      org.omg.CORBA.Objectref=ootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);
      // get the root naming context
      // NameService invokes the name service
      org.omg.CORBA.Object objRef =
      orb.resolve_initial_references("NameService");
      // Use NamingContextExt which is part of the Interoperable
      // Naming Service (INS) specification.
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      // bind the Object Reference in Naming
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);
      System.out.println("HelloServer ready and waiting ...");
      // wait for invocations from clients
      orb.run();
    }
        catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }
```

Formatado: Inglês (E.U.A.)

```
      System.out.println("HelloServer Exiting ...");
          }
}
//----
```

HelloClient.java

```
// Copyright and License Sun Microsystems.
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class HelloClient
{
  static Hello helloImpl;
  public static void main(String args[])
    {
      try{
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);
        // get the root naming context
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
         // resolve the Object Reference in Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
        System.out.println("Obtained a handle on server object: " +
helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();
       } catch (Exception e) { System.out.println("ERROR : " + e) ;
         e.printStackTrace(System.out);
         }
    } }
```

Formatado: Norueguês (Bokmål)

```
//-------------
```

Hello.idl

```
//Copyright and License Sun Microsystems.
module HelloApp
{  interface Hello
  {
  string sayHello();
  oneway void shutdown();
  };
};
```

Formatado: Norueguês (Bokmål)

Apêndice B - Tabelas com os resultados obtidos

Tabela 11 - Resultados obtidos com o Navegador Internet Explorer 6.0. – Retângulo.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de uso de arquivo de Página Kb.
HtmlVazio	1		61,46	31,16	0,57	1	17756	5323	9020	9128
Bmp-16	385	1024x768	78,13	36,9	0,76	1	18536	5554	9496	9552
Bmp-256	770	1024x768	85,31	43,81	1,11	3,77	18944	5666	9904	9940
Bmp-24	2305	1024x768	87	44,31	2,94	5,72	20480	6346	11432	11524
JPG	23	1024x768	83,7	39,93	0,58	1	20464	6020	11428	11488
Gif	5	1024x768	82,22	38,65	0,63	1,5	18888	5615	9896	9896
Png	5	1024x768	87,64	41,12	1,49	2,66	20464	6013	11416	11484
SVG	1	1024x768	70,29	40,36	2,94	1	24852	7229	14700	14700

Excluído: *11*

Inserido: *11*

Excluído: *10*

Inserido: *10*

Excluído: *11*

Formatado: Espaçamento entre linhas: simples

Tabela 12 - Resultados obtidos com o Navegador Internet Explorer 6.0 – Círculos.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio	1		61,46	31,16	0,57	1	17756	5323	9020	9128
Bmp-16	385	1024x768	75,692	33,80	1,11	1	18536	5577	9488	9584
Bmp-256	770	1024x768	74,25	35,35	1,51	3,85	18928	5684	9776	9868
Bmp-24	3.073	1024x768	92	46,45	4,29	9,63	21208	6430	12180	12288
JPG	75	1024x768	78,62	35,25	1,05	2	20456	6032	11316	11424
Gif	80	1024x768	74,85	33,75	0,87	1,5	18896	5631	9868	9932
Png	308	1024x768	86,38	40,02	2,15	2	20572	6117	11440	11784
SVG	19	1024x768	92,13	50,81	3,23	1	25648	7489	15532	15532

Excluído: *12*

Inserido: *12*

Excluído: *11*

Inserido: *11*

Excluído: *12*

Formatado: Espaçamento entre linhas: simples

Tabela 13 - Resultados obtidos com o Navegador Firefox 1.5 – Retângulo.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio	1		67,71	29,18	0,29	1	27628	8641	22436	23900
Bmp-16	385	1024x768	87,46	38,68	0,54	1,63	28516	9146	16828	20640
Bmp-256	770	1024x768	93,05	40,30	0,99	2,4	28324	8864	16960	20696
Bmp-24	2305	1024x768	99,52	49,71	2,93	5,26	28288	9319	15472	19216
JPG	23	1024x768	86,72	39,83	0,32	1	30136	9349	16972	20804
Gif	5	1024x768	80,77	36,33	0,13	1	30172	9128	17136	20908
Png	5	1024x768	86,62	40,07	0,35	1,8	32264	9885	20824	22888
SVG	1	1024x768	74,61	32,33	0,53	1	27984	9278	17156	18680

Excluído: 13
Inserido: 13
Excluído: 12
Inserido: 12
Excluído: 13
Formatado: Espaçamento entre linhas: simples

Tabela 14 - Resultados obtidos com o Navegador Firefox 1.5 – Círculos.

Excluído: *14*
Inserido: *14*
Excluído: *13*
Inserido: *13*
Excluído: *14*
Formatado: Espaçamento entre linhas: simples

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio	1		67,71	29,18	0,29	1	27628	8641	22436	23900
Bmp-16	385	1024x768	89,46	39,00	0,86	2,33	30336	8994	17012	21036
Bmp-256	770	1024x768	92,01	37,91	1,19	2	28968	9213	18368	22128
Bmp-24	3.073	1024x768	94,72	49,25	4,07	7,90	28292	8955	16992	19020
JPG	75	1024x768	83,2	36,515	0,68	1	30412	8953	17160	21188
Gif	80	1024x768	85,34	37,19	0,58	0,66	30376	8934	17040	21076
Png	308	1024x768	89,87	39,01	1,121	1	31892	9859	20568	22860
SVG	19	1024x768	100	64,46	1,68	1	28556	9037	22316	25112

Tabela 15 - Resultados obtidos com o Navegador Opera 9.02.- Retângulo.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio	1		49,27	23,37	0,33	1	19464	9883	17112	17520
Bmp-16	385	1024x768	96,55	35,38	2,15	2	19420	10342	17140	17548
Bmp-256	770	1024x768	98,46	46,02	3,45	3,22	19448	12487	17112	17520
Bmp-24	2305	1024x768	100	62,21	9,30	2,25	19348	8941	17240	17460
JPG	23	1024x768	82	32,20	1,18	1	20140	9432	17912	18088
Gif	5	1024x768	69,7	27,78	0,85	1	25420	14605	20244	23364
Png	5	1024x768	84,36	34,55	1,45	1	23284	10591	19948	21260
SVG	1	1024x768	78,31	30,24	1,05	1	19464	9883	17112	17520

Excluído: *15*

Inserido: *15*

Excluído: *14*

Inserido: *14*

Excluído: *15*

Formatado: Espaçamento entre linhas: simples

Tabela 16 - Resultados obtidos com o Navegador Opera 9.02.- Círculos.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio	1		49,27	23,37	0,33	1	19464	9883	17112	17520
Bmp-16	385	1024x768	96,65	38,30	2,38	1,83	19472	9863	17248	17488
Bmp-256	770	1024x768	99	49,56	3,79	1,35	19488	10022	17236	17472
Bmp-24	3.073	1024x768	100	66,90	12,07	2,31	19492	13025	17228	17468
JPG	75	1024x768	81,19	33,09	1,54	1	19264	9438	17184	17444
Gif	80	1024x768	77,42	31,26	1,33	1	20112	9479	17932	17976
Png	308	1024x768	96,18	41,49	2,39	1,61	23084	12647	20408	21780
SVG	19	1024x768	97,45	39,32	2,19	1	23148	11080	19844	21160

Excluído: *16*

Inserido: *16*

Excluído: *15*

Inserido: *15*

Excluído: *16*

Formatado: Espaçamento entre linhas: simples

Tabela 17 - Resultados obtidos em Garça - Navegador Opera 9.02.-Retângulo.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio	1		35	15,69	2,40	1	19892	9900	17212	17932
Bmp-16	385	1024x768	40	10,75	31,9	5	19764	10924	17037	17132
Bmp-256	770	1024x768	46	8,40	62,9	4	19940	13952	16902	16906
Bmp-24	2305	1024x768	38	7,54	114,5	6	19988	10024	17730	17732
JPG	23	1024x768	75,42	20,8	2,25	3	20368	10876	17101	17101
Gif	5	1024x768	68,34	21,76	1,43	3	25248	15784	19807	19900
Png	5	1024x768	46	18,38	7,14	4	23100	11176	19663	20056
SVG	1	1024x768	68	26,32	1,83	1	20396	22592	17412	17412

Excluído: *17*

Inserido: *17*

Excluído: *16*

Inserido: *16*

Excluído: *17*

Formatado: Espaçamento entre linhas: simples

Tabela 18 - Resultados obtidos em Garça - Navegador Opera 9.02. - Círculos.

Imagens	Kb	Dimensões	Pico Máx. Ocupação do Processador RDTSC	% Médio de Ocupação Processador	Tempo Médio de Aquisição Seg.	% Médio Ocupação Processador Servidor	Pico de Ocupação da Memória Kb.	Falta de Página	Uso de arquivo de Página Kb.	Pico de Uso de arquivo de Página Kb.
HtmlVazio			59	28,75	0,58	1	19892	9900	17212	17932
Bmp-16	385	1024x768	43	19,96	31,9	4	20172	10163	17848	18008
Bmp-256	770	1024x768	42	17,86	62,9	6	20188	10442	17946	17972
Bmp-24	3.073	1024x768	46	11,10	114,5	8	20292	13625	17824	17868
JPG	75	1024x768	75	28,18	0,651	3	20264	10138	17782	17782
Gif	80	1024x768	80	30,58	0,831	2	20224	10279	18334	18376
Png	308	1024x768	49	19,39	7,141	4	23584	13147	21108	21108
SVG	19	1024x768	100	60,36	2,13	2	23148	11280	20444	20444

Excluído: *18*

Inserido: *18*

Excluído: *17*

Inserido: *17*

Excluído: *18*

Formatado: Espaçamento entre linhas: simples

Apêndice C - Código para identificar a velocidade da CPU (CPUID)

```
// identifica se o processador fornece suporte ao RDTSC e mede a velocidade
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <windows.h>
struct no
{  int info;
   struct no *prox;
} *lista, *p, *q;
void cria()
{  lista=NULL;
}
void insere_lista(int val)
{ // p= (struct no*) malloc( 1*sizeof(struct no));
  p=(struct no*) calloc (1,sizeof(struct no));
  p->info=val;
  p->prox=NULL;
  if (lista==NULL)
         lista=p;
  else
  {     q=lista;
        while (q->prox!=NULL)
        q=q->prox;
        q->prox=p;
  }
}
void grava()
{
  char str[30];
  p=lista;
  FILE *fp;
  fp = fopen("EDW.txt","a+"); //cria arquivo EDW.txt
  while (p!=NULL)
  {itoa(p->info,str,10);
   fputs(str,fp);
   fputs(";",fp);
   p=p->prox;
  }
```

Formatado: Italiano (Itália)

Formatado: Italiano (Itália)

Formatado: Italiano (Itália)

Formatado: Francês (França)

Formatado: Francês (França)

```
  fclose(fp);
}//insere_lista(sys);//grava();
void cpuid()
{
char *strFeatures[2]={"RDTSC - Read Time Stamp Counter","PSN - Processor
Serial Number"};
char FabricanteID[13];
unsigned long VersaoInfo;
unsigned long FabricInfo;
      _asm
      {     mov eax,0
            cpuid                                    //instrucao CPUID
            mov dword ptr [FabricanteID],ebx
            mov dword ptr [FabricanteID+4],edx
            mov dword ptr [FabricanteID+8],ecx
            mov byte  ptr [FabricanteID+12],0
      }
//Mostra o fabricante"
      printf("Processador Identificado: %s \n ", FabricanteID);
      _asm
      {     mov eax,1
            cpuid
            mov VersaoInfo,eax
            mov FabricInfo,edx
      }
printf("Processador possui suporte a:\n");
      for(long a=0;a<2;a++)
      {     _asm xor ebx,ebx
            _asm mov ebx,a
            _asm bt FabricInfo,ebx
            printf("%s\n",strFeatures[a]);
     }
      return ;
}
DWORD veloc()
{
      LARGE_INTEGER ulFreq, ulTicks, ulValue, ulStartCounter, ulEAX_EDX;
      if (QueryPerformanceFrequency(&ulFreq))
      {
 QueryPerformanceCounter(&ulTicks);
 ulValue.QuadPart = ulTicks.QuadPart + ulFreq.QuadPart;
```

```
 __asm RDTSC //instução  read time stamp counter
 __asm mov ulEAX_EDX.LowPart, EAX//salva os valores em EAX
 __asm mov ulEAX_EDX.HighPart, EDX
 ulStartCounter.QuadPart = ulEAX_EDX.QuadPart;//armazena os valores
 // laço de um segundo - medir a qtde de ciclos nesse intervalo
 do {
      QueryPerformanceCounter(&ulTicks);
 } while (ulTicks.QuadPart <= ulValue.QuadPart);
  __asm RDTSC
  __asm mov ulEAX_EDX.LowPart, EAX
  __asm mov ulEAX_EDX.HighPart, EDX
  return   (DWORD)   ((ulEAX_EDX.QuadPart   -   ulStartCounter.QuadPart)   /
1000000);
      }else {
    return 0;
 }
int escolha(void);
escolha(void)
{ char s[2];
  int c;
  printf("1. Identificar CPU\n");
  printf("2. Velocidade\n");
  printf("3. Sair\n");
  do {
    printf("\n<--Entre com sua escolha-->: ");
    gets(s);
    c =atoi(s);
  } while(c<0 || c>3);
  return c;
}
void main(void)
{
for(;;) {
 switch(escolha()) {
    case 1: cpuid();
      break;
    case 2:  DWORD veloc();
                 grava();
             {
      DWORD RDTSC;
      RDTSC = veloc();
```

```
        insere_lista(RDTSC);
        grava();
 printf("Frequencia da CPU : %u MHz\n", RDTSC);
          }
      break;
    case 3: exit(0);
      }
  }
}
```

Apêndice D - Código para coletar consumo da CPU

```
#include <stdafx.h>
#include "CPU.h"
#include <ctype.h>
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <tchar.h>
#include <windows.h>
#include <iomanip>
#include <stdio.h>
#include <time.h>
using namespace std;
struct no
{  int info;
   struct no *prox;
} *lista, *p, *q;
void cria()
{  lista=NULL;
} void insere_lista(int val)
{ p=(struct no*) calloc (1,sizeof(struct no));
  p->info=val;
  p->prox=NULL;
  if (lista==NULL) lista=p;
  else
  {  q=lista;
     while (q->prox!=NULL)
     q=q->prox; q->prox=p;
  }
```

Formatado: Italiano (Itália)

Formatado: Francês (França)

```
}
void grava()
{ char str[30];
  p=lista; FILE *fp;
  fp = fopen("EDW.txt","a+"); //cria arquivo EDW.txt
  while (p!=NULL)
  {     itoa(p->info,str,10);
        fputs(str,fp);
        fputs(";",fp);
        p=p->prox;
  }  fclose(fp);
}
int _tmain(int argc, _TCHAR* argv[])
{   CPU cpu;
    int sys;
    int process;
    TKTime upTime;
    while (!kbhit())
    {   Sleep( 600 );
        process = cpu.GetUsage( &sys, &upTime );
        wprintf(  _T("Processo  :  %d\t    :  %d\n"),  sys,  upTime);//uptime
//é o frame do getticker
        insere_lista(sys);
   }    grava();
        return 0;
 }
cpu.cpp
#include "StdAfx.h"
#include "SmartLock.h"
TKLong  CPU::s_time;
TKDelay CPU::s_delay;
int    CPU::s_count = 0;
int    CPU::s_index = 0;
TKLong CPU::s_idleTime;
TKLong CPU::s_tempoNucleo;
TKLong CPU::s_tempoUsuario;
int    CPU::s_lastCpu = 0;
int    CPU::s_cpu[];
TKLong CPU::s_tempoNucleoProcess;
TKLong CPU::s_tempoUsuarioProcess;  //userTime
int    CPU::s_lastCpuProcess;
```

Formatado: Francês (França)

Formatado: Francês (França)

Formatado: Italiano (Itália)

```
int    CPU::s_cpuProcess[];
TKLong CPU::s_lastUpTime = 0;
HINSTANCE CPU::s_hKernel = NULL;
pfnGetSystemTimes CPU::s_pfnGetSystemTimes = NULL;
CPU::CPU()
{    ::InitializeCriticalSection( &m_lock );
    if( s_hKernel == NULL )
    {   s_hKernel = LoadLibrary( _T("Kernel32.dll") );
        if( s_hKernel != NULL )
        {    s_pfnGetSystemTimes    =    (pfnGetSystemTimes)GetProcAddress(
s_hKernel, "GetSystemTimes" );
            if( s_pfnGetSystemTimes == NULL )
            { FreeLibrary( s_hKernel ); s_hKernel = NULL;
            }
        }
    }  s_delay.Mark();
}CPU::~CPU()
{   if( s_hKernel == NULL )
    { FreeLibrary( s_hKernel ); s_hKernel = NULL;
    } ::DeleteCriticalSection( &m_lock );
}int CPU::GetUsage( int* pSystemUsage, TKTime* pUpTime )
{   TKLong sTime;
    int sLastCpu;
    int sLastCpuProcess;
    TKTime sLastUpTime;
    {   SmartLock lock( &m_lock );
        sTime           = s_time;
        sLastCpu        = s_lastCpu;
        sLastCpuProcess = s_lastCpuProcess;
        sLastUpTime     = s_lastUpTime;
    }    if( s_delay.MSec() <= 100 )//100
    {    if( pSystemUsage != NULL )
            *pSystemUsage = sLastCpu;
        if( pUpTime != NULL )
            *pUpTime = sLastUpTime;
        return sLastCpuProcess;
    } TKLong time;//estrutura de tempo
      TKLong idleTime;
    TKLong tempoNucleo;  //kernelTime
    TKLong tempoUsuario;
    TKLong tempoNucleoProcess;
```

Formatado: Norueguês (Bokmål)

```
    TKLong tempoUsuarioProcess;
    GetSystemTimeAsFileTime( (LPFILETIME)&time );
    if( sTime == 0 )
    {  // for the system
        if( s_pfnGetSystemTimes != NULL )
        {
           s_pfnGetSystemTimes(                      (LPFILETIME)&idleTime,
(LPFILETIME)&tempoNucleo, (LPFILETIME)&tempoUsuario );
        }   else
        {   idleTime    = 0;
            tempoNucleo  = 0;
            tempoUsuario    = 0;
        }
        {   FILETIME createTime;
            FILETIME exitTime;
            GetProcessTimes( GetCurrentProcess(), &createTime, &exitTime,
           (LPFILETIME)&tempoNucleoProcess,
            LPFILETIME)&tempoUsuarioProcess );
        }
        {   SmartLock lock( &m_lock );
            s_time                  = time;
            s_idleTime              = idleTime;
            s_tempoNucleo           = tempoNucleo;
            s_tempoUsuario          = tempoUsuario;
            s_tempoNucleoProcess    = tempoNucleoProcess;
            s_tempoUsuarioProcess   = tempoUsuarioProcess;
            s_lastCpu           = 0;
            s_lastCpuProcess    = 0;
            s_lastUpTime        = tempoNucleo + tempoUsuario;
            sLastCpu        = s_lastCpu;
            sLastCpuProcess = s_lastCpuProcess;
            sLastUpTime     = s_lastUpTime;
        }
        if( pSystemUsage != NULL )
            *pSystemUsage = sLastCpu;
        if( pUpTime != NULL )
            *pUpTime = sLastUpTime;
        s_delay.Mark();
        return sLastCpuProcess;
    }
    TKLong div = ( time - sTime );
```

```
    if( s_pfnGetSystemTimes != NULL )
    {s_pfnGetSystemTimes((LPFILETIME)&idleTime,    (LPFILETIME)&tempoNucleo,
(LPFILETIME)&tempoUsuario );
    }
    else
    {   idleTime    = 0;
        tempoNucleo  = 0;
        tempoUsuario    = 0;
    } {   FILETIME createTime;
        FILETIME exitTime;
        GetProcessTimes( GetCurrentProcess(), &createTime, &exitTime,
                         (LPFILETIME)&tempoNucleoProcess,
                         (LPFILETIME)&tempoUsuarioProcess );
    } int cpu;
      int cpuProcess;
    { SmartLock lock( &m_lock );
      TKLong usr = tempoUsuario   - s_tempoUsuario;// clockticks do usuario
        TKLong ker = tempoNucleo - s_tempoNucleo;//clockticks do kernel
        TKLong idl = idleTime - s_idleTime;//clockticks da interrupcao
        TKLong  sys  =  (usr  +  ker);  //clockticks  do  sistema  =  usuario  +
kernel
        if( sys == 0 )
            cpu = 0;
        else
            cpu = int( ((sys - idl) *100) / sys ); // calcula o percentual
de uso da CPU
        cpuProcess = int( ( ( ( tempoUsuarioProcess - s_tempoUsuarioProcess
) + ( tempoNucleoProcess - s_tempoNucleoProcess ) ) *100 ) / div );
        s_time                  = time;
        s_idleTime              = idleTime;
        s_tempoNucleo           = tempoNucleo;
        s_tempoUsuario          = tempoUsuario;
        s_tempoNucleoProcess    = tempoNucleoProcess;
        s_tempoUsuarioProcess   = tempoUsuarioProcess;
        s_cpu[(s_index++) %5]   = cpu;
        s_cpuProcess[(s_index++) %5] = cpuProcess;
        s_count ++;
        if( s_count > 5 )
            s_count = 5;
            int i;
        cpu = 0;
        for( i = 0; i < s_count; i++ )
```

```
            cpu += s_cpu[i];
                cpuProcess = 0;
        for( i = 0; i < s_count; i++ )
            cpuProcess += s_cpuProcess[i];
        cpu         /= s_count;
        cpuProcess  /= s_count;
                s_lastCpu        = cpu;//
        s_lastCpuProcess = cpuProcess;
        s_lastUpTime     = tempoNucleo + tempoUsuario;
        sLastCpu         = s_lastCpu;
        sLastCpuProcess  = s_lastCpuProcess;
        sLastUpTime      = s_lastUpTime;
    }
    (_T("CPU:%d  sys:%d div %d"), cpuProcess, cpu, div );
        if( pSystemUsage != NULL )
        *pSystemUsage = sLastCpu;
    if( pUpTime != NULL )
        *pUpTime = sLastUpTime;
    s_delay.Mark();
    return sLastCpuProcess;
}
```

Apêndice E - Código do script para medir o tempo de Aquisição

```
var T1=new Date();
var TempoCarga=null;
var T2=null;
function pageLoaded()
{ T2=new Date();
  TempoCarga=(0.001*(T2.getTime()-T1.getTime()));
  alert("Tempo de Carregamento: " + TempoCarga + " segundos");
}
```

Apêndice F - Código do ícone em SVG

```
<!DOCTYPE svg PUBLIC '-//W3C//DTD SVG 20001102//EN'
'http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd'>
<svg width='100%' height='100%' xmlns='http://www.w3.org/2000/svg'
   onload='Iniciio(evt)'
```

```
   onmousedown='Grab(evt)'
   onmousemove='Drag(evt)'
   onmouseup='Drop(evt)'>
   <title>Arrastando Icones</title>
   <desc> </desc>
   <script><![CDATA[
      var DocumentoSVG = null;
      var SVGPRI = null;
      var CoordInic = null;
      var BackDrop = null;
      var Posicao = null;
      function Iniciio(evt)
       { DocumentoSVG = evt.target.ownerDocument;
         SVGPRI = DocumentoSVG.documentElement;
         CoordInic = SVGPRI.createSVGPoint();
         GrabPoint = SVGPRI.createSVGPoint();
         BackDrop = DocumentoSVG.getElementById('BackDrop');
      }  function Grab(evt)
      {  var targetElement = evt.target;
         if ( BackDrop != targetElement )
         {  Posicao = targetElement;
            Posicao.parentNode.appendChild( Posicao );
            Posicao.setAttributeNS(null, 'pointer-events', 'none');
            var TransfCoord = Posicao.getCTM();
            GrabPoint.x = CoordInic.x - Number(TransfCoord.e);
            GrabPoint.y = CoordInic.y - Number(TransfCoord.f);
         }
      };  function Drag(evt)
      {  GetCoordInic(evt);
         if (Posicao)
         { var newX = CoordInic.x - GrabPoint.x;
           var newY = CoordInic.y - GrabPoint.y;
           Posicao.setAttributeNS(null, 'transform', 'translate(' + newX +
','  + newY + ')');
         }
      };
      function Drop(evt)
      { if ( Posicao )
         {var targetElement = evt.target;
            Posicao.setAttributeNS(null, 'pointer-events', 'all');
            if ( 'Folder' == targetElement.parentNode.id )
```

```
            {
            }
            else
            {
            }
            Posicao = null;
         }
      };
function GetCoordInic(evt)
      { var newScale = SVGPRI.currentScale;
       var translation = SVGPRI.currentTranslate;
       CoordInic.x = (evt.clientX - translation.x)/newScale;
       CoordInic.y = (evt.clientY - translation.y)/newScale;
      };
      ]]></script>
       // aqui eh definido o tamanho da tela
      <rect  id='BackDrop'  x='-5%'  y='-5%'  width='110%'  height='110%'
fill='none' pointer-events='all' />
   //desenha o retangulo
   <a xlink:href=http://galileu.fundanet.br/wdc/ target="_blank" id="a1549"
    transform="translate(1.81772,128.7923)">
    <rect       width="80"       x="18"       height="20"       y="-100"
style="opacity:0.5;fill:#0000ff;stroke:#ff0000;stroke-width:5"
id="rect1551" />
   </a> <a xlink:href=http://www.google.com.br/ target="_blank" id="a1549"
transform="translate(5.81772,128.7923)">
     <circle  cx="40"  cy="-30"  r="20"  stroke="red"  stroke-width="5"
fill="green"/> </a>
  <a
xlink:href="ftp://mirror.aarnet.edu.au/pub/eudora/servers/unix/popper/"
target="_blank"
     id="a1560"  transform="translate(5.81772,128.7923)">  <circle  cx="40"
cy="20" r="20" stroke="black"
     stroke-width="7" fill="yellow"/>//preenchimento externo e interno
 </a> <g id='Folder'> </g>
</svg>
```

Formatado: Português

Formatado: Português

Apêndice G - Código dos WIMP em SVG

```
//códigos botões e radio botton
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" [
    <!ATTLIST svg
              xmlns:gui CDATA #IMPLIED
              xmlns:a3 CDATA #IMPLIED
              a3:scriptImplementation CDATA #IMPLIED>
    <!ATTLIST script
              a3:scriptImplementation CDATA #IMPLIED>
]>
<svg onload="init(evt)"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:a3="http://ns.adobe.com/AdobeSVGViewerExtensions/3.0/"
     a3:scriptImplementation="Adobe">
    <script type="text/ecmascript" a3:scriptImplementation="Adobe"
            xlink:href="Button.js"/>
    <script                                          type="text/ecmascript"
a3:scriptImplementation="Adobe"><![CDATA[
        var note;
        var canvas;
        var last_down  = null;
        var last_radio = null;
        var notes = new Array(
            "Abrir","Ligar","Desligar","Fechar","Alarme","Reset","Sem
Funcao","Fechar"      );
          function init(e) {
            if ( window.svgDocument == null )
                svgDocument = e.target.ownerDocument;
            note   = svgDocument.getElementById("note").firstChild;
            canvas = svgDocument.getElementById("canvas");
            var bw = 18;
            for ( var y = 0; y < 2; y++) {//qtde de botoes
                for ( var x = 0; x < 6; x++ ) {
                    var left = x * (bw + 3) + 85 //posicao horizontal
                    var top   = y * (bw + 3) + 35// posicao vertical dos
botoes
                    var b = new Button(
```

Formatado: Fonte: (Padrão) Courier New, 10 pt

Formatado: Fonte: (Padrão) Courier New, 10 pt

Excluído: ¶
¶

Formatado: Fonte: (Padrão) Times New Roman, 14 pt, Português (Brasil)

Formatado: Português

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

```
                        left, top, say_hello,
                          "up", "down",
                          svgDocument.documentElement
                      );
                      b.index = y * 2 + x;
                  }
              }
              new Button(150, 180, toggle, "check_off", "check_on", canvas);
              new    Button(10,    162,    radio,    "radio_off",    "radio_on",
canvas);//8, 162,
              new Button(10, 180, radio, "radio_off", "radio_on", canvas);
          }
          function get_reference(ref) {
              var entity = svgDocument.createEntityReference(ref);
              var value  = entity.firstChild.data;
              return value;
          }
          function say_hello(button) {
              if ( last_down ) {
                  last_down.set_select(false);
              }
              button.set_select(true);
              last_down = button;
              note.data = notes[button.index];
          }
          function toggle(button) {
              button.set_select(!button.selected);
          }
          function radio(button) {
              if ( last_radio ) {
                  last_radio.set_select(false);
              }
              button.set_select(true);
              last_radio = button;
          }
      ]]></script>
      <defs> <g id="up">
              <rect   x="1"   y="1"   rx="3"   ry="3"   width="18"   height="18"
fill="black"/> //retangulos dos botoes
 <rect x="-1" y="-1" rx="3" ry="3" width="18" height="18" fill="white"/>
<rect rx="3" ry="3" width="18" height="18" fill="rgb(155,155,155)"/>
```

Formatado: Inglês (E.U.A.)

```
        </g>    <g    id="down">    <rect    rx="3"    ry="3"    width="18"
height="18"stroke-width="1"stroke="rgb(100,100,100)"
fill="rgb(128,128,128)"/> </g>
            <g id="radio_off">
            <circle    r="5"    stroke-width="1"    stroke="rgb(100,100,100)"
fill="white"/>
        </g> <g id="radio_on"> //tamanho do circulo do radio Button
            <circle    r="8"    stroke-width="1"    stroke="rgb(100,100,100)"
fill="white"/> <circle r="5" fill="black"/>
        </g>   <g   id="check_off"><rect   width="10"   height="10"   stroke-
width="1" stroke="rgb(100,100,100)" fill="white"/> </g>
        <g  id="check_on">  <rect  width="10"  height="10"  stroke-width="2"
stroke="rgb(100,100,100)" fill="white"/>
            <line x1="1" y1="1" x2="9" y2="9" stroke="black"/>//diagonal do
check box
            <line x1="1" y1="9" x2="9" y2="1" stroke="black"/>//diagonal do
check box  </g> </defs>
<g  id="canvas"  transform="translate(48,5)"><text  id="note"  x="95"  y="95"
text-anchor="middle">Nao Selecionado</text>
<g  font-size="10pt"><text  x="165"  y="189">Check  Box!</text>//posicao  do
label do check box </g>
<g font-size="10pt">
            <text  x="20"  y="165">Radio  Button!</text>/posicao  do  radio
button
 </g> </g></svg>
//////////////////////
//menu em SVG//
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE       svg       PUBLIC       "-//W3C//DTD       SVG       1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-basic.dtd">
<svg>
<script type="text/javascript">
<![CDATA[
var redVal=0;
var greenVal=0;
var blueVal=0;
function changeDescriptionText(evt,siteNum)
{  targetText=svgDocument.getElementById("siteDescription");
   var targetButton = evt.getTarget();
   targetButton.setAttribute("fill","url(#rainbowRad)");
   if(siteNum==1){
      var    newDescriptionText    =    svgDocument.createTextNode("Link    SVG
servidor 1! ");
   }  else if(siteNum==2){
```

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

Formatado: Português

Formatado: Inglês (E.U.A.)

Formatado: Inglês (E.U.A.)

```
      var   newDescriptionText   =   svgDocument.createTextNode("Link   SVG
servidor 2!");
   }   else if(siteNum==3){
      var   newDescriptionText   =   svgDocument.createTextNode("Link   SVG
servidor 3!.");
   }
targetText.replaceChild(newDescriptionText,targetText.getFirstChild());
}
function resetText()
{     targetText=svgDocument.getElementById("siteDescription");
      var  resetDescriptionText  =  svgDocument.createTextNode("Selecione  o
site para Navegar!                          ");
      targetText.replaceChild(resetDescriptionText,targetText.getFirstChild
());  var targetButton = evt.getTarget();
////////////////////
// Ponteiros em SVG
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE       svg       PUBLIC       "-//W3C//DTD       SVG       1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-basic.dtd">
<svg  version="1.1"  baseProfile="full"  xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"     id="svg-root"     width="60%"
height="60%" viewBox="0 0 480 360">
      <SVGponteiro
xmlns="http://www.w3.org/2000/02/svg/testesuite/description/">
      </SVGponteiro><title id="testee-title">Interacao c/ Ponteiros</title>
      <desc id="teste-desc"> </desc> <g id="teste-body-content">
      <g class="button"> <g cursor="crosshair">//cursor tipo cruz
<rect stroke="none" fill="silver" x="148" y="50" width="80" height="25" />
<text  stroke="none"  fill="black"  font-family="Geneva,  Arial,  Helvetica,
sans-serif" font-weight="bold" font-size="16" x="170" y="68">Cruz</text>
</g><g  cursor="default">  <rect  stroke="none"  fill="silver"  x="148"  y="80"
width="80" height="25" />
<text  stroke="none"  fill="black"  font-family="Geneva,  Arial,  Helvetica,
sans-serif" font-weight="bold" font-size="16" x="160" y="97">Padrao</text>
                  </g>
                  <g cursor="pointer">
                        <rect  stroke="none"  fill="silver"  x="148"  y="110"
width="80" height="25" />
<text  stroke="none"  fill="black"  font-family="Geneva,  Arial,  Helvetica,
sans-serif" font-weight="bold" font-size="16" x="170" y="127">Link</text>
                  </g>
                  <g cursor="move">//cursor mover
                        <rect  stroke="none"  fill="silver"  x="148"  y="140"
width="80" height="25" />
```

Formatado: Inglês (E.U.A.)

Formatado: Português

```
                    <text      stroke="none"      fill="black"      font-
family="Geneva,  Arial,  Helvetica,  sans-serif"  font-weight="bold"  font-
size="16" x="165" y="158">Mover</text>
                </g> <g cursor="text">//cursor selecionar texto
                    <rect  stroke="none"  fill="silver"  x="252"  y="20"
width="80"    height="25"/><text    stroke="none"    fill="black"    font-
family="Geneva,  Arial,  Helvetica,  sans-serif"  font-weight="bold"  font-
size="16" x="270" y="38">Texto</text>
                </g> <g cursor="wait">// cursor ampulheta
                    <rect  stroke="none"  fill="silver"  x="252"  y="50"
width="80" height="25" />
                    <text      stroke="none"      fill="black"      font-
family="Geneva,  Arial,  Helvetica,  sans-serif"  font-weight="bold"  font-
size="16" x="265" y="68">Espera</text>
                </g><g cursor="help">// cursor interrogação
                    <rect  stroke="none"  fill="silver"  x="252"  y="80"
width="80" height="25" />
                    <text      stroke="none"      fill="black"      font-
family="Geneva,  Arial,  Helvetica,  sans-serif"  font-weight="bold"  font-
size="16" x="270" y="98">Ajuda</text>
                </g>  <g cursor="url(#magglass),crosshair">
                    <rect  stroke="none"  fill="silver"  x="252"  y="110"
width="80"    height="25"/><text    stroke="none"    fill="black"    font-
family="Geneva,  Arial,  Helvetica,  sans-serif"  font-weight="bold"  font-
size="16"        x="270"        y="128">&lt;url&gt;</text><rectstroke="none"
fill="silver" x="252" y="140" width="80" height="25"/>
                </g> </g>
            <g      stroke="black"      stroke-width="1"      fill="#900">
<rect  stroke="none"  fill="silver"  x="148"  y="20"  width="80"  height="25"
cursor="w-resize"/>  <text  stroke="none"  fill="black"  font-family="Geneva,
Arial,  Helvetica,  sans-serif"  font-weight="bold"  font-size="16"  x="162"
y="40">Resize</text><rect   stroke="none"   fill="silver"   x="252"   y="140"
width="80"                  height="25"                  cursor="s-resize"/>
<text  stroke="none"  fill="black"  font-family="Geneva,  Arial,  Helvetica,
sans-serif"   font-weight="bold"   font-size="16"   x="260"   y="160">Resize
S</text>
            </g>  </g>
          </svg>
```

Formatado: Português

Formatado: Português

Formatado: Português

Formatado: Português

Página 8: [12] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [13] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [14] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [15] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [16] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [17] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [18] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 8: [19] Excluído **DesrtEagle** **28/9/2007 5:11**

Página 12: [20] Excluído **DesrtEagle** **29/9/2007 1:35**

I.E 6.0	% Pico Máx. CPU	% Consumo Médio CPU	Tempo Médio Aquisição - Seg.	% Ocupação CPU Servidor	Pico Ocupação Memória
Htmlvazio	61,46	31,16	0,57	1	19.464K
SVG Ret.	70,29	40,36	2,94	1	24.852K
SVG Círc.	92,13	50,81	3,23	1	25.648K

Firefox 1.5	**% Pico Máx. CPU**	**% Consumo Médio CPU**	**Tempo Médio Aquisição - Seg.**	**% Ocupação CPU Servidor**	**Pico Ocupação Memória**
Htmlvazio	67,71	29,18	0,29	1	27.628K
SVG Ret.	74,61	32,33	0,53	1	27.984K
SVG Círc.	100	64,46	1,68.	1	28.556K
Opera 9.02	**% Pico Máx. CPU**	**% Consumo Médio CPU**	**Tempo Médio Aquisição - Seg.**	**% Ocupação CPU Servidor**	**Pico Ocupação Memória**
Htmlvazio	49,27	23,37	0,33	1	19.464K
SVG Ret.	78,31	30,24	1,05	1	19.464K
SVG Círc.	97,45	39,32	2,19	1	23.148K