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

ANDRÉ LUIZ DA SILVA FREIRE

SISTEMAS DISTRIBUÍDOS EM AMBIENTES GRÁFICOS SVG NA WEB SEMÂNTICA NO MODELO MVC

Marília
2006

ANDRÉ LUIZ DA SILVA FREIRE

SISTEMAS DISTRIBUÍDOS EM AMBIENTES GRÁFICOS NA WEB SEMÂNTICA NO MODELO MVC

Dissertação apresentada ao Programa de Mestrado 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 Sistemas Computacionais).

Orientador:
Prof. Dr. Marcos Luiz Mucheroni

Marília
2006

ANDRÉ LUIZ DA SILVA FREIRE

SISTEMAS DISTRIBUÍDOS EM AMBIENTES GRÁFICOS NA WEB SEMÂNTICA NO MODELO MVC

Banca examinadora da dissertação apresentada ao Programa de Mestrado do UNIVEM,/F.E.E.S.R., para obtenção do Título de Mestre em Ciência da Computação. Área de Concentração: Arquitetura de Sistemas Computacionais.

Resultado: _____

ORIENTADOR: Prof. Dr. Marcos Luiz Mucheroni

1º EXAMINADOR (a):_____

2º EXAMINADOR(a):_____

Marília, 7 de Julho de 2006.

DEDICATÓRIA

Aos meus pais que sempre me apoiaram principalmente nas horas mais difíceis.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus e a meus familiares.

A todos que participaram deste momento tão importante da minha vida.

Ao meu orientador Marcos Luiz Mucheroni por sempre esta disponível e pronto a me orientar mesmo nos momentos quase que impossíveis.

FREIRE, André Luiz S. **Sistemas Distribuídos em ambientes gráficos na web semântica no modelo MVC.** 2006. 104 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

Este trabalho tem como finalidade oferecer um sistema para que um usuário sem conhecimento específico de alguma linguagem consiga construir objetos e gráficos via Internet utilizando a linguagem SVG. O Sistema apresentado foi desenvolvido baseado no modelo MVC. O Objetivo principal é apresentar um sistema baseado nas tecnologias de construção JSP, Servlets, Beans e SVG, o que caracteriza a construção de um ambiente para uma nova web, a web Semântica. Não se trata da construção de um sistema gráfico para web, mas um sistema que ofereça a possibilidade de se trabalhar com gráficos de conteúdo e significado, ou seja, usando os conceitos da web semântica, no formato SVG. Também foram estudadas a tecnologia JSP, que juntamente com o servidor Tomcat, permitem que páginas web tenham um conteúdo dinâmico, ainda permitindo a personalização das páginas por cada usuário. É feita uma análise dos *web Services* que solucionam alguns problemas de comunicação entre sistemas com arquiteturas e ambientes heterogêneos, ampliando a interação que os usuários possam ter na sua relação com a Internet na atualidade.

Palavra-chave: Web Semântica, Serviços na web, SVG.

FREIRE, André Luiz S. **Sistemas Distribuídos em ambientes gráficos na web semântica no modelo MVC.** 2006. 104 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

This work has as purpose the construction of a system for object development and graphs language SVG saw Internet using. This system was developed based in model MVC. The main objective is to present a system based on the technologies of construction JSP, Servlets, Beans and SVG, what it characterizes the construction of an environment for new web, web Semantics. One is not about the construction of a graphical system for web, but a system that offers the possibility of working with content graphs and meaning, that is, using the concepts of web semantics, in format SVG. Also it had been studied technology JSP, that together with the Tomcat server, still allows that pages web have a dynamic content, allowing the personalization of the pages for each user. An analysis of web Services is made having allowed to identify that it is possible the heterogeneous communication between systems with architectures and environments, extending the interaction that the users can have in its relation with the Internet in the present time.

Keywords: Distributed System, Semantic web, web Services, SVG.

SUMÁRIO

Listas de Figuras e Tabelas

INTRODUÇÃO

Este trabalho tem como objetivo principal fazer um estudo geral sobre sistemas distribuídos na web, serviços na web e o formato gráfico *Scalable Vector Graphics* (SVG), com a finalidade de apresentar um ambiente de trabalho distribuído para geração de objetos e gráficos em SVG.

O ambiente de trabalho apresenta a sua construção baseada em um modelo de arquitetura para desenvolvimento de sistemas web chamado MVC (*Model, View and Controler*, ou Modelo Visualização e Controlador) utilizando-se das seguintes tecnologias: SVG, JSP, Servlets, Java e JavaBeans.

O ambiente visa apresentar um sistema no qual o usuário não precisa se preocupar com códigos nem ter conhecimento algum da linguagem que esta sendo utilizada para a construção dos objetos ou gráficos no formato SVG.

Também foi realizada uma pesquisa sobre as tecnologias *eXtensible Markup Language* (XML) e a linguagem de programação Java, além de Servlets e *JavaServer Pages* (JSP) como linguagens de desenvolvimento dinâmico para a web.

O trabalho aqui apresentado surgiu principalmente para atender a necessidade, na atualidade, de se oferecer um ambiente gráfico para a web, que possa disponibilizar uma nova web para os usuários, uma web de significado. Que além de um ambiente gráfico com qualidade possa também oferecer serviços com a capacidade de usufruir toda as qualidades das tecnologias aqui utilizadas.

O ambiente gráfico distribuído, que ora é apresentado, não é apenas mais um ambiente gráfico para a Internet, mas é um ambiente gráfico distribuído de forma vetorial e com um conteúdo para uma nova web, a web semântica, que busca tornar o ambiente interoperável, de melhor definição de resolução e com melhor desempenho.

No capítulo 2, desenvolvem-se os sistemas distribuídos com seus principais conceitos e características, fazendo uma breve introdução do seu surgimento até os dias atuais e sua evolução, principalmente para a web. Isto porque, a limitação de sistemas simples de computação, a necessidade de melhoramento da velocidade de processamento e, sobretudo o desempenho das máquinas levou-se ao melhor desenvolvimento dos sistemas distribuídos.

No capítulo 3 é apresentado o *web Services*, como uma nova tecnologia para a comunicação na web entre sistemas heterogêneos. Essa nova tecnologia, com seu conceito atualizado no que diz respeito às ferramentas XML, *Web Service Definition Language* (WSDL), *Universal Description Discovery and Integration* (UDDI) e do protocolo *Simple Object Access Protocol* (SOAP), figuram ilustrando o funcionamento de um *web Service*.

No capitulo 4 são apresentadas as tecnologias que foram utilizadas para a construção do sistema baseado no modelo MVC, além de exemplos de cada uma destas tecnologias. Também são observados dois modelos novos de construção para a web baseados no MVC que são o HMVC e MMVC.

No capítulo 5, é apresentado o formato SVG, uma linguagem gráfica totalmente baseada no XML. Trata-se de uma maneira geral a principal tecnologia da proposta deste trabalho. Pode-se afirmar que um conceito simplificado para o SVG é o de uma linguagem para a construção de gráficos em duas dimensões e também para a descrição de aplicações gráficas em XML. Da mesma forma, são analisadas as suas principais qualidades, áreas de atuações, empresas envolvidas com o desenvolvimento do SVG, assim como, suas fundamentais características de linguagem e de conteúdo.

Ainda no capítulo 6 é apresentado o modelo desenvolvido por este trabalho, detalhando o seu funcionamento e o ilustrando com figuras.

Nas conclusões são apresentadas as dificuldades encontradas para o desenvolvimento deste trabalho, assim como as vantagens e resultados obtidos da pesquisa que pode ajudar significativamente na ampliação da interação que as pessoas passam a ter com a Internet na atualidade, propiciando às mesmas um maior aproveitamento das novas tecnologias.

CAPÍTULO 2 - SISTEMAS DISTRIBUÍDOS

Os sistemas distribuídos têm se afirmado a partir da influência de vários fatores. Como por exemplo, a evolução das tecnologias computacionais que diminuem o custo do hardware e melhoram cada vez mais a construção e desenvolvimento de softwares, a descentralização das informações antes totalmente isoladas em computadores e a distribuição no processamento de dados e informações, contribuindo assim para a introdução do mesmo em diversas áreas da computação.

Diante da limitação de sistemas simples de computação, da necessidade de melhoramento da velocidade de processamento e também do fraco desempenho das máquinas com necessidades crescentes, criou-se a necessidade de sistemas que possibilitassem uma resposta cada vez mais rápida a tarefas a serem executada por processadores (computadores).

Para resolver estes problemas pode-se citar como primeiro grande avanço o desenvolvimento de microprocessadores, que possibilitaram o processamento mais rápido e a diminuição no tamanho e custo dos computadores (Mainframes). Em seguida, o desenvolvimento das redes locais que possibilitaram a descentralização das informações e a construção de sistemas baseados em mais de um processador (TANENBAUM, 1992).

Pode-se destacar os sistemas paralelos, baseados em múltiplos processadores, como um dos primeiros sistemas criado para o processamento mais rápido das tarefas. Em seguida, as redes de computadores e por fim os sistemas distribuídos.

Uma das grandes diferenças entre as redes de computadores já existentes e os sistemas distribuídos é a transparência, ou seja, em um sistema distribuído o usuário enxerga o sistema inteiro como se fosse apenas uma única máquina, enquanto que, em uma rede de computadores o usuário enxerga todas as máquinas do sistema, além dos

recursos e informações de cada uma.

Pode-se citar vantagens dos sistemas distribuídos em relação aos sistemas centralizados, como: economia no custo-benefício em relação aos mainframes, velocidade, crescimento e aumento de outros dispositivos, compartilhamento de dados, compartilhamento de dispositivos, comunicação entre os computadores e flexibilidade da distribuição de tarefas.

Como desvantagens pode-se destacar o pouco conhecimento por se tratar de uma nova tecnologia, uma pequena quantidade de linguagens para desenvolvimento que efetivamente explorem a concorrência e o paralelismo pela distribuição de processos, o congestionamento da rede devido ao fluxo de informações e, como também, a dificuldade de se gerenciar e fazer a segurança deste tipo de sistema.

O ambiente de software que se propõe neste trabalho busca usufruir todas as vantagens apresentadas por sistemas distribuídos para aplicações na web e diminuir algumas das desvantagens, além de abstrair do usuário final toda a lógica de desenvolvimento do sistema.

2.1 - Conceitos para a arquitetura de sistemas distribuídos para redes e web

As primeiras redes de computadores tinham uma visão centralizada do computador, tanto que, um único computador era o responsável por todos os serviços e recursos da rede.

A Figura 2.1 ilustra como são as redes centralizadas, com um único computador disponibilizando os serviços aos outros computadores da rede.

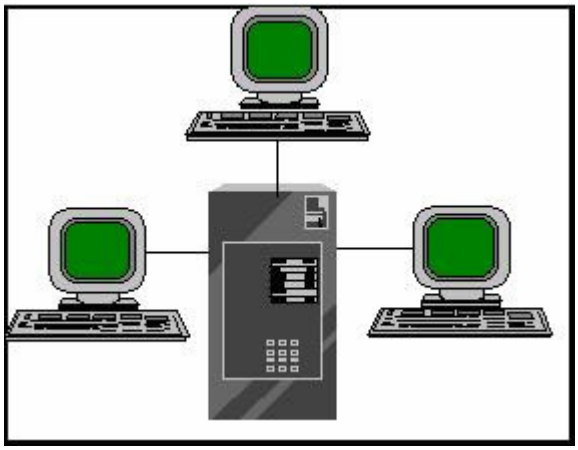

Figura 2 1 - Redes centralizada

A evolução das redes de computadores possibilitou a redução de custos de comunicação e hardware, aumentando consideravelmente o crescimento destas redes.

Com o crescimento das redes e da web, os sistemas distribuídos podem ser vistos de uma nova forma.

Esta forma torna-se cada vez mais um grande desafio para programadores e ***webdesigners***, que buscam uma solução para seus problemas e os problemas da web.

O sistema proposto neste trabalho baseia-se em um modelo que busca resolver alguns problemas de desenvolvimento de software para a web, este modelo ou forma será apresentado mais à frente neste trabalho (modelo MVC).

Com a necessidade cada vez maior de atender as necessidades dos usuários por requisições de serviços e recursos pela web, precisou-se ampliar a distribuição e o compartilhamento destes recursos, assim sendo, computadores passaram a ser interligados para que juntos pudessem disponibilizar um número maior de benefícios.

Com isso cresceram os estudos para o compartilhamento de recursos, que podem ser classificados de duas maneiras: explícito e implícito.

O compartilhamento explícito de recursos na rede oferece aos usuários a possibilidade de acessarem e manipularem diretamente os recursos remotos, sendo que

o usuário deverá ter o conhecimento necessário dos recursos remotos, podendo dificultar o gerenciamento da rede.

Por outro lado, nos compartilhamentos implícitos de recurso, o acesso aos recursos remotos é feito automaticamente, de forma transparente ao usuário, desta maneira, em grande parte os sistemas de recursos implícitos, acabaram se tornando nas características básicas dos sistemas distribuídos.

O termo sistemas distribuídos muitas vezes é confundido por outros termos parecidos como processamento distribuído, computação distribuída e processamento de dados distribuídos. Na década de 70, foi convencionado que muitos elementos de um sistema poderiam ser distribuídos. Por isso, a principal característica de um sistema distribuído é a descentralização do controle de dispositivos.

Um sistema distribuído é formado por um conjunto de módulos, compostos por pelo menos processador e memória, interligados fracamente através de um subsistema de comunicação de topologia arbitrária. Este *hardware* deve oferecer facilidades de comunicação entre processadores e entre processos, os quais, cooperando sob um controle descentralizado, possibilitam a execução de programas de aplicação (ECKHOUSE, 1978).

O modelo de computação centralizada é abandonado visto que nenhum processador poderá coordenar os outros, mas todos deverão cooperar em harmonia, como acontece em uma comunidade (PEEBLES & MANNING, 1978).

Um Sistema Distribuído é uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente, note-se a abrangência desta definição, que inclui aplicações web (TANENBAUM, 1992).

Uma grande vantagem encontrada nos sistemas distribuídos é sem dúvida a independência da localização geográfica, podendo existir tanto em um ambiente de uma

rede local quanto em redes distantes fisicamente, porém, uma das maiores dificuldades é o desenvolvimento de software para sistemas distribuídos para a web ou mesmo *desktop*.

Algumas características dos sistemas distribuídos que os fazem serem citados para a construção de muitos projetos são apresentadas por (PETERSON, 1979).

a) crescimento incremental: em um sistema distribuído existe a possibilidade de um crescimento em expansão, possibilitando um sistema maior.

b) confiabilidade: com a multiplicidade de componentes e certos graus de autonomia de suas partes, os sistemas distribuídos são potencialmente mais confiáveis.

c) estrutura: os sistemas distribuídos refletem a estrutura organizacional a qual servem.

d) proteção: a proteção se dá muito mais pela distribuição lógica do que pela distribuição física.

2.2 - A Arquitetura Cliente-Servidor

Inicialmente o termo Cliente-Servidor era aplicado para a arquitetura de software que descrevia o processamento entre dois programas, sendo que a aplicação cliente requisitava um serviço para ser executado pelo programa servidor, que poderia ou não estar na mesma máquina. Hoje, quando se fala em Cliente-Servidor, refere-se a um processamento cooperativo distribuído, onde o relacionamento, entre clientes e servidores, é feito entre componentes tanto de software quanto de hardware.

Portanto, a arquitetura Cliente-Servidor envolve duas ou mais máquinas em um processo cooperativo para execução de uma aplicação.

A Arquitetura Cliente-Servidor se caracteriza em ser um sistema distribuído, que no início da década de 90 veio contrapor a arquitetura centralizada e a de computadores isolados.

Com a *Local Area Network* (LAN), os arquivos e programas mais importantes começaram a residir em um servidor de arquivos, protegidos por senhas e backups automáticos.

A formação de uma arquitetura Cliente-Servidor é um sistema de computação que oferece três componentes básicos para o compartilhamento de recursos: computador cliente, computador servidor e um meio para conectá-los, a rede.

Um dos meios para a comunicação Cliente-Servidor é o protocolo de transporte *Transmission Control Protocol* (TCP) que funciona verificando os dados que são enviados de forma correta, na seqüência apropriada e sem erros.

Um computador servidor é aquele que disponibiliza serviços ou armazena dados para serem consultados e compartilhados por outros computadores, enquanto que, um computador cliente é aquele que busca por serviços ou dados compartilhados em um servidor.

A Figura 2.2 mostra uma rede de computadores com serviços oferecidos pelos mesmos.

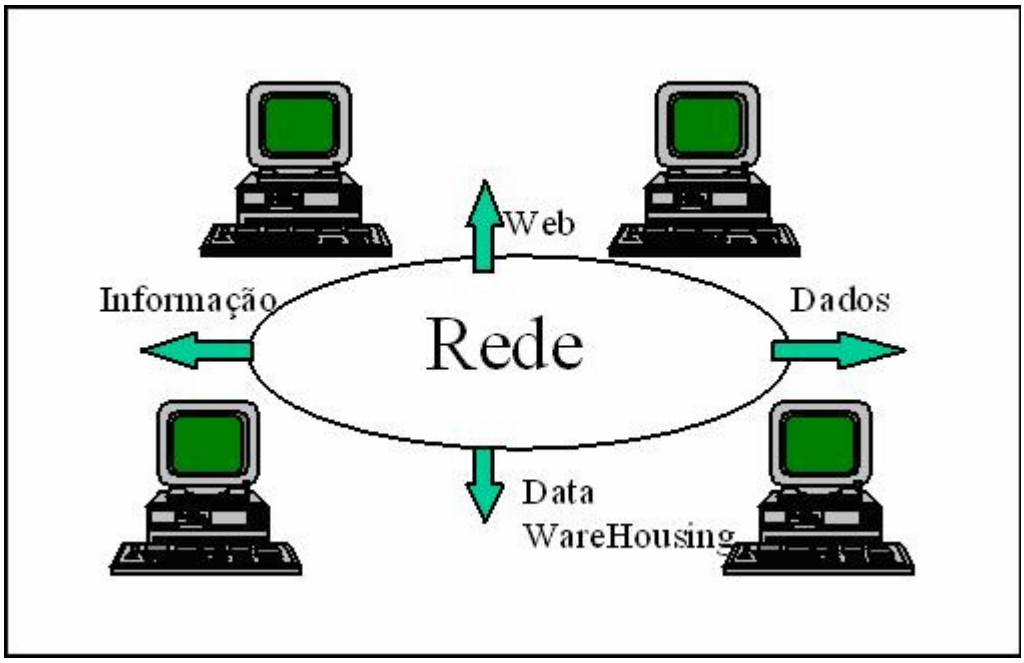

Figura 2 2 - Sistema com computadores oferecendo serviços

Na Figura 2.2 pode-se perceber que existe uma infinidade de serviços que podem ser oferecidos por computadores servidores, permitindo-se assim o crescimento do ambiente computacional de forma flexível e escalar, disponibilizando novos recursos e serviços ao usuário final com certa transparência.

A arquitetura Cliente-Servidor ainda pode ter diversas configurações como servidores de arquivos, servidores de banco de dados, servidores de web, entre outros. Todas essas divisões garantem um melhor gerenciamento e facilidade de manutenção dos serviços devido a sua concentração em diferentes locais.

No servidor *World Wide Web* (WWW) ou servidor web, requisições são efetuadas pelos clientes através da Internet por um navegador e tratadas em servidor específico, que cuida da identificação e envio dos arquivos requisitados.

A Figura 2.3 apresenta a arquitetura Cliente-Servidor, mostrando servidores que compartilham serviços quando os computadores clientes os solicitam.

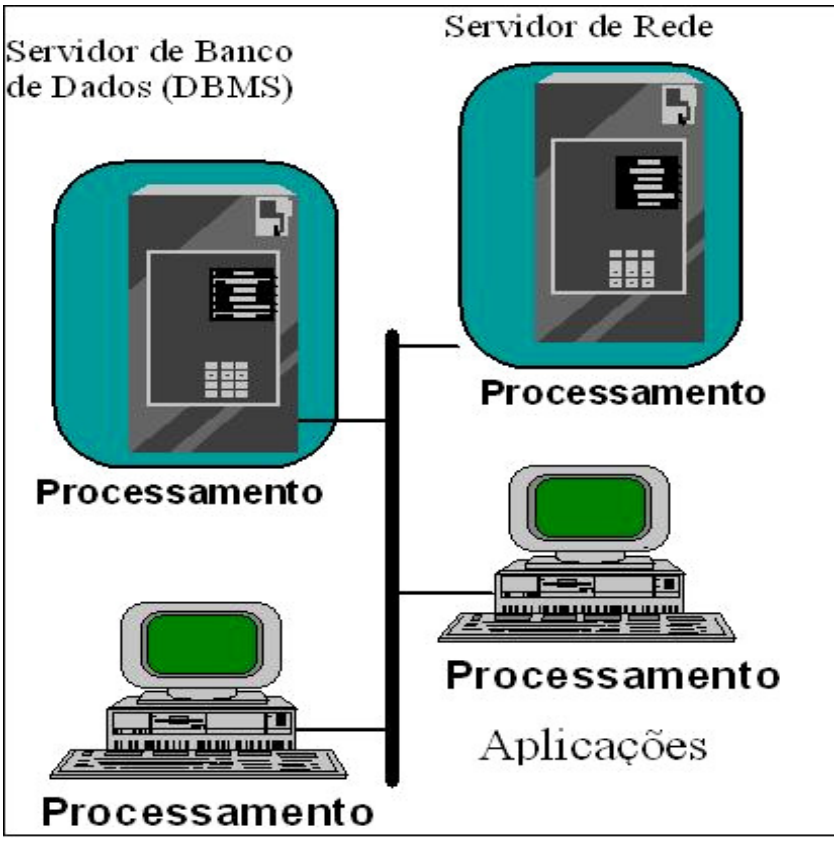

Figura 2 3 - Servidores compartilhando serviços com clientes

Pode-se citar algumas facilidades e algumas dificuldades encontradas até aqui na construção de sistemas distribuídos Cliente-Servidor como: A) a possibilidade da construção de grandes ou pequenas redes em uma empresa com a customização de *hardware* e o beneficiamento da utilização dos recursos compartilhados. B) Utilização de aplicações de multimídia e gráficas na rede, que permite aos usuários uma maior expectativa e, conseqüentemente, uma maior produtividade. C) Com a colaboração do balanceamento dividido por duas ou mais máquinas diminuem-se o tempo de espera e o tráfego na rede. D) Para evitar que um servidor seja bastante solicitado, são necessários uma boa distribuição e gerenciamento do processamento, além de uma excelente lógica da aplicação. E) Aplicações distribuídas são muito mais complexas de serem construídas devido a um grande número de parâmetros a serem definidos,

desconhecimento da tecnologia e a falta de ferramentas adequadas, influenciando assim no grande tempo de demora para a construção destes sistemas.

Para efeitos de estudo sistemas Cliente-Servidor são aqueles que compartilham recursos e serviços, para um melhor desempenho da rede e uma maior otimização das máquinas. No tópico a seguir encontra-se uma segunda geração dos sistemas Cliente-Servidor. Neste será descrito um pouco mais dos novos serviços como o servidor web.

2.3 - Internet e o Servidor web como sistema Cliente-Servidor

A Internet foi criada a partir da necessidade de computadores fisicamente distantes pudessem transferir dados e informações entre si.

Inicialmente apenas como uma rede privada surgida pelo ***Advanced Research Projects Agency Network*** (ARPANET). Conseqüentemente, havendo a possibilidade de troca de informações e serviços, iniciou-se a difundir está rede também para universidades, centros de pesquisas e entre outros, passando a ser chamada de Arpanet.

A grande quantidade de redes que começaram a surgir e a interconectar-se se tornou o que chamamos de Internet, ou seja, um conjunto de redes que trocam informações entre si, através de servidores que disponibilizam serviços e de clientes que buscam informações.

Com a Internet surgiram novidades como:

- Novo enfoque para a relação Cliente-Servidor.
- Através de formulários HTML clientes e servidores podem trocar informações, acessarem bancos de dados e informações ficam à disposição dos clientes.
- Incompatibilidade entre os tipos de dados vem sendo ainda a principal dificuldade Cliente-Servidor de empresas que utilizam diferentes tipos de programas e linguagens.

Com a evolução dos sistemas distribuídos não apenas começa a existir a distribuição de recursos entre dois computadores como uma estação de trabalho e um servidor de banco de dados, mas também novos recursos e serviços começam a serem distribuídos e novos elementos começam a aparecer no sistema.

Servidores começam a surgir com a finalidade de retirar das estações de trabalho grandes partes do processamento que os mesmos realizam. Os principais servidores a fazerem este trabalho são os servidores de aplicação e servidores web.

Um servidor de aplicação, por exemplo, é o responsável por retirar o restante da camada de manipulação de dados que fica na estação cliente, além de concentrar a lógica de negócio, antes feita pela estação cliente e um servidor de banco, restando apenas para a estação cliente o processamento visual com o usuário, ficando mais leve e exigindo uma menor configuração, além de melhorar o desempenho.

Já os servidores web vão mais longe ainda, permitindo retirar das estações de trabalho até a parte da lógica da interface visual, deixando-as responsáveis apenas por interpretar o código "HTML" enviado pelos servidores. Porém, com a utilização de componentes como Java e ActiveX, parte do processamento pode retornar à estação de trabalho.

Com certeza todo este trabalho traz ainda maior complexidade na implementação dos mesmos.

2.4 - Servidor web Apache Jakarta Tomcat

O Servidor web Apache Jakarta Tomcat foi desenvolvido pela Sun Microsystems, juntamente com a Java Community Process e a Apache Jakarta Project para ser um *servlet container* capaz de possibilitar o uso de Servlets e do JavaServer Pages como linguagens dinâmicas para desenvolvimento de páginas na web.

O Apache Jakarta Tomcat é um servidor web específico para páginas JSP e Servlets, podendo também trabalhar em conjunto com outros servidores web, possibilitando assim direcionar todo o conteúdo solicitado que não seja de uma página JSP, como por exemplo, um conteúdo de uma página HTML a outro servidor, ficando responsável apenas pelas páginas JSP solicitadas.

Na Figura 2.4 é possível ver o esquema de funcionamento do servidor Tomcat trabalhando sozinho, ou seja, recebendo tanto solicitações em busca de páginas JSP, como solicitações de conteúdos diferentes de outras páginas da web. As solicitações são feitas a partir de um navegador e através do protocolo HTTP, responsável pelo envio de uma solicitação e pelo recebimento da mesma.

Figura 2 4 - Esquema de funcionamento do servidor Tomcat

Para efeitos de otimização do serviço o melhor seria ter o servidor web Tomcat trabalhando em conjunto com um outro servidor web HTTP. Isto possibilitaria que solicitações que não fossem do tipo JSP pudessem ser mais rapidamente respondidas e o servidor Tomcat receberia apenas solicitações JSP, otimizando o serviço.

A Figura 2.5 mostra o esquema de funcionamento em conjunto dos dois servidores.

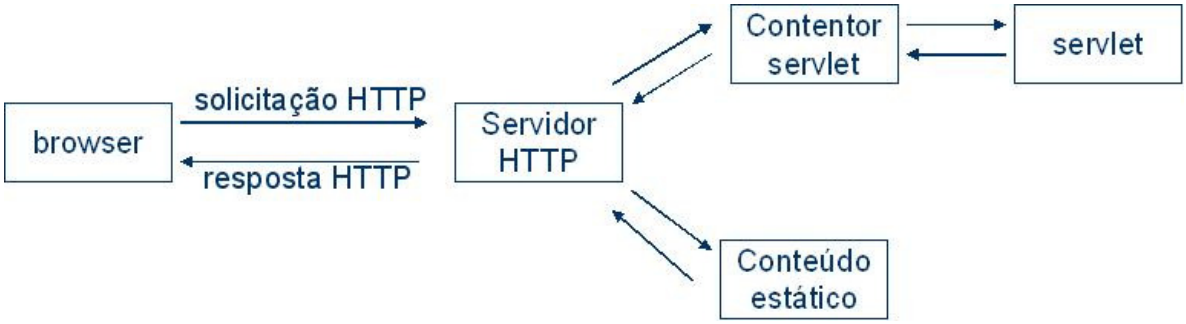

Figura 2 5 - Esquema de funcionamento do servidor Tomcat em conjunto com outro servidor.

Para a configuração e funcionamento do Tomcat, deve-se ter instalado no computador servidor o Java SDK e configurar a variável de sistema JAVA_HOME com o endereço da pasta onde está instalado o Java SDK. Por exemplo, caso seja instalado o Java SDK na pasta adequada contendo a linguagem Java, logo, a variável de sistema JAVA_HOME deve conter este endereço.

Para verificar o correto funcionamento deve-se inicializar o arquivo startup, localizada dentro da pasta onde foi descompactado o Tomcat do computador servidor. Por exemplo, c:\Tomcat\bin.

Caso não aconteça nenhum erro o Tomcat será inicializado com sucesso, bastando agora acessar o seu navegador e digitar o endereço do IP na porta adequada, que será mostrada a página de inicialização do Tomcat. O Tomcat vem configurado com a porta de acesso 8080, que depois pode ser modificada.

Para o desenvolvimento de um sistema que utilize os *Enterprise JavaBeans* (EJB), ou seja, são JavaBeans distribuídos, é necessário utilizar um outro servidor web, como por exemplo o JBOSS, pois, o Tomcat não é adequado para EJB.

Com a Internet surgiram vários serviços que podem ser oferecidos por servidores a clientes, dentro desses serviços pode-se destacar como uma das novas funcionalidades da web para interação entre sistemas que não compartilham a mesma linguagem e estrutura os web Services.

No próximo capítulo são apresentados os serviços web e a comunicação entre web Services, assim como sua arquitetura e componentes.

CAPÍTULO 3 - SERVIÇOS WEB

Neste capítulo são desenvolvidos os conceitos de Serviços na web, em especial, web Services e sua arquitetura de funcionamento, assim como o protocolo SOAP, o registro UDDI, além da linguagem XML e da linguagem de interface WSDL.

Apresenta-se os princípios básicos de uma arquitetura voltada para serviços web e as tecnologias padrão que estão surgindo sob coordenação do W3C e que suportarão a implementação desta arquitetura.

Com a Internet foi natural o surgimento de vários serviços para a utilização e aproximação cada vez mais de usuários finais, além da possibilidade de crescimento e faturamento por empresas de comercio eletrônico, hospedagem e desenvolvimento de *sites* e entre outras, que buscam cada vez mais um sistema seguro, estável e eficiente. Os usuários esperam facilidade, rapidez e garantia de recebimento e um bom negócio efetuado. Certamente serviços foram criados, muitos deles sem uma estrutura ou regra a ser seguida, por isso, muitos dos serviços que se tem hoje são incompatíveis e não conseguem se comunicar, que é uma das principais dificuldades encontrada por estes serviços hoje, por isso, web Services buscam aproximar cada vez mais sistemas e facilitar a comunicação entre os mesmos.

O trabalho busca utilizar tecnologias que buscam desde a sua construção já apresentar os sistemas preparados para uma comunicação com outros. Tecnologia essas como o SVG, apresentado no capitulo 5 em detalhes, baseado no XML, que é umas tecnologias utilizadas para a interoperabilidade entre sistemas incompatíveis.

A interoperabilidade entre bases de dados heterogêneas e distribuídas, e ainda entre linguagens e ambientes é provida pela web semântica.

Um dos principais problemas encontrados na Internet em relação aos sistemas disponíveis é justamente a comunicação entre os mesmos, por isso, busca-se a integração dos serviços web disponíveis.

3.1 – Definições básicas sobre web Services

Cada vez mais as informações em empresas, escolas, órgãos públicos e quaisquer outros organismos sociais tornam-se de extrema importância para a tomada de decisões. Por isso, em uma empresa, sistemas financeiros, de venda, contábil e principalmente sistemas construídos para a web precisam interoperar, ou seja, necessitam da troca de informações.

A principal dificuldade encontrada hoje entre sistemas que necessitam de informações contidas em outros sistemas é justamente a necessidade de interoperação entre estes sistemas. Na maioria das vezes são sistemas heterogêneos, ou seja, são sistemas diferentes por serem construídos com linguagens diferentes, em que cada uma possui a sua forma de definir dados e variáveis, tornando-se incompatíveis quando tentam se comunicar.

Grande parte dos sistemas ainda são construídos isoladamente, ou seja, não são projetados para compartilhar informações ou trocar informações/dados, o que num futuro próximo pode-se tornar um grande problema na disponibilização das informações para um outro sistema, além de um grande custo financeiro.

Web Services surgem com a missão de resolver este problema, tanto o financeiro como o do tempo, facilitando a comunicação entre sistemas de uma empresa, mas não só isso, surge como uma solução para sistemas da web, que não necessitam estar no mesmo espaço físico, mas podem estar em qualquer lugar, desde que estejam conectados à Internet.

Web Services é um serviço para que sistemas possam trocar informações entre si, além de ser uma forma simples, flexível e padronizada de utilizar a malha da Internet para uma integração de sistemas heterogêneos, ou seja, Web Services conseguem a interoperar os sistemas independentemente de plataforma, de linguagem, de sistemas operacionais e até mesmo entre dispositivos diferentes (CHAMPION, 2002).

Outras tecnologias como o CORBA, DCOM e RMI também oferecem a possibilidade de sistemas distribuídos poderem trocar informações entre si, porém, não utilizam padrões abertos como, por exemplo, o XML, que facilitam a padronização da comunicação entre sistemas.

Para tal, diferentemente do DCOM e CORBA, os Web Services são baseados em padrões abertos como: XML, SOAP, HTTP (*Hyper Text Markup Language*), UDDI e WSDL, padrões amplamente utilizados atualmente. Os Web Services utilizam um mecanismo de comunicação baseado em XML, que passa a ser então uma espécie de "Linguagem Universal para troca de dados e mensagens entre aplicações".

Por serem os Web Services baseados no XML, os mesmos conseguem prover de formas estruturadas de formatação de dados e padronização de processos para descrever a maneira como estes serviços são realizados, ou seja, cada informação tem um significado e é identificada (CHAMPION, 2002).

Os Web Services têm como proposta permitir que aplicações possam se interagir uma com as outras sem a intervenção das pessoas.

Os formatos utilizados para a construção de um Web Service são apresentados a seguir (XML, WSDL, SOAP e UDDI).

3.2 – O Formato XML

O XML pode ser considerado um padrão que tem como objetivo o intercâmbio de dados. Permite que sistemas possam trocar informações de uma forma mais flexível

que arquivos textos, já que se pode dar significado aos dados que estão sendo manipulados.

O formato XML nasce a partir do formato SGML, com a finalidade de resolver os problemas da grande quantidade de informações eletrônicas existentes, dando significado a quantidade imensa de dados existentes na web (QUIN, 2003).

Da mesma forma que o HTML, o XML utiliza comandos para a representação dos dados, porém, existem diferenças entre esses dois padrões. O HTML especifica como os dados devem ser exibidos, enquanto que, o XML se preocupa com o significado dos dados armazenados.

A seguir um exemplo simples utilizando o XML.

```
<curso>
   <aluno>
      <nome>joao</nome>
      <media>7.9</media>
   </aluno>
   <aluno>
      <nome>maria</nome>
      <media>8.6</media>
   </aluno>
</curso>
```

A partir do exemplo acima é possível notar que os comandos utilizados indicam a estrutura e o conteúdo armazenado. Além disso, os comandos XML são extensíveis, ou seja, os documentos XML representam fielmente os dados por ele armazenados, enquanto que, o HTML possui um conjunto de comandos já pré-definidos e fixos, não permitido a adição de novos conjuntos de comandos.

Cada estrutura do documento XML é chamada: elemento. O exemplo apresentado possui os seguintes elementos: curso, aluno, nome e media. Elementos localizados dentro de outros elementos são chamados sub-elementos. Por exemplo, no código apresentado, aluno é um sub-elemento de curso e nome e média são sub-elementos de aluno.

Um documento XML deve seguir uma estrutura para ser válido. A linguagem para a definição desta estrutura é chamada *Document Type Definition* (DTD).

O DTD define os comandos que serão utilizados num documento XML e sua estrutura hierárquica, incluindo a ordem que devem aparecer. No exemplo abaixo se pode observar um DTD para o documento XML definido anteriormente.

```
<!ELEMENT curso (aluno)+>
<!ELEMENT aluno (nome, media)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT media (#PCDATA)>
```

Para o DTD acima, o elemento curso é o de mais alto nível, indica também que entre os comandos <curso> e </curso> do exemplo XML visto anteriormente deve aparecer pelo menos um aluno. A segunda linha indica que cada elemento aluno deve conter os elementos nome e media. E a terceira e a quarta linha indicam que os elementos nome e media armazenarão dados.

O DTD é o que torna o documento XML portável, se uma aplicação recebe um documento XML baseado no DTD aluno, esta pode processar o documento de acordo com as regras especificas do DTD. Outra característica que torna o XML portável é o fato do documento não incluir instruções de como os dados serão mostrados. Manter os dados separados das instruções de formatação faz com que os dados possam ser exibidos em diferentes mídias. Além disso, os documentos XML se apresentam no formato texto, permitindo assim que o documento seja lido tanto por pessoas como por máquinas, facilitando a sua interpretação.

3.3 – A Interface WSDL

WSDL é um arquivo XML que facilita a conexão com os serviços oferecidos pelos Web Services, descrevendo as regras para se conhecer as funcionalidades específicas e as informações corretas que devem ser passadas para os web Services, ou

seja, os tipos de dados e os protocolos que podem ser utilizados para o envio e o recebimento de mensagens, possibilitando assim a sua utilização.

Os Serviços do WSDL são definidos com pontos de acesso na rede (*endpoints*).

Um documento WSDL contém os seguintes elementos: *types* container para definição de tipos, *message* (definição dos tipos de dados que são passados nas operações), *operation* (descrição de uma ação disponibilizada pelo serviço), *port Type* (conjunto de operações suportadas no *endpoints*), *Binding* (especificação do formato dos dados para um determidado *portType*), port (endpoint) e *service* (conjunto de endpoints).

3.4 – O Protocolo SOAP

O XML não é suficiente para a troca de informações na web, é necessário um protocolo que possa enviar e receber mensagens contendo o XML. Um desses protocolo é o SOAP.

SOAP é o protocolo de comunicação dos Web Services, pois, descreve o que está contido na mensagem e que tipo de negociação pode ser realizado, além de definir se a informação é opcional ou obrigatória.

> O *HyperText Transfer Protocol* (HTTP) é o protocolo de servidores de página HTML universalmente utilizado, sendo o Apache-Tomcat um servidor que implementa também este protocolo, construído sob as mesmas bases que o Netscape e o Explorer.

O propósito do SOAP é o de fazer chamada a procedimentos remotos em cima do protocolo HTTP, sendo um protocolo para troca de informações em um ambiente distribuído e é composto de três partes: *envelope*, *header* e *body*.

3.5 – O Formato UDDI

O *Universal Description, Discovery and Integration* (UDDI) é um protocolo que tem como função descobrir Web Services e ajudar na sua publicação através de um *service provider*.

UDDI consiste em um diretório onde ficam armazenadas informações sobre os Web Services e também de quem os construiu, possibilitando assim, que os mesmos possam ser encontrados, apenas se fazendo uma consulta ao UDDI.

3.6 – Tecnologias para a criação de Web Service

Existem algumas tecnologias capazes de criarem Web Services, como por exemplo: a tecnologia .Net, que é um produto da Microsoft, que oferece um ambiente chamado .Net Framework para a construção de Web Service, além de oferecer também uma integração com outras tecnologias e linguagens de terceiros.

Uma outra tecnologia para a construção de web Services seria a utilização da linguagem Java que disponibiliza uma API que utiliza o pacote *Java web Services Developer Pack* (JWSDP) da SUN Microsystems.

3.7 - Funcionamento e arquitetura de um web Service

A principal finalidade de um Web Service é disponibilizar os serviços e comunicações automaticamente entre sistemas, sem a intervenção de pessoas, sendo assim, agilizando as transações entre sistemas.

Um Web Service pode ser solicitado via protocolo HTTP, utilizando-se o protocolo SOAP para a comunicação. A solicitação, então, é processada e enviada de volta a quem a solicitou. Além disso, é preciso ter definida a interface WSDL e publicar o serviço em um registro UDDI, que fica disponível em um *service provider*.

Na Figura 3.1 é apresentado o funcionamento básico de uma solicitação para um Web Service, que oferece seu serviço através de um *service provider* e apresenta suas características e serviços através do WSDL.

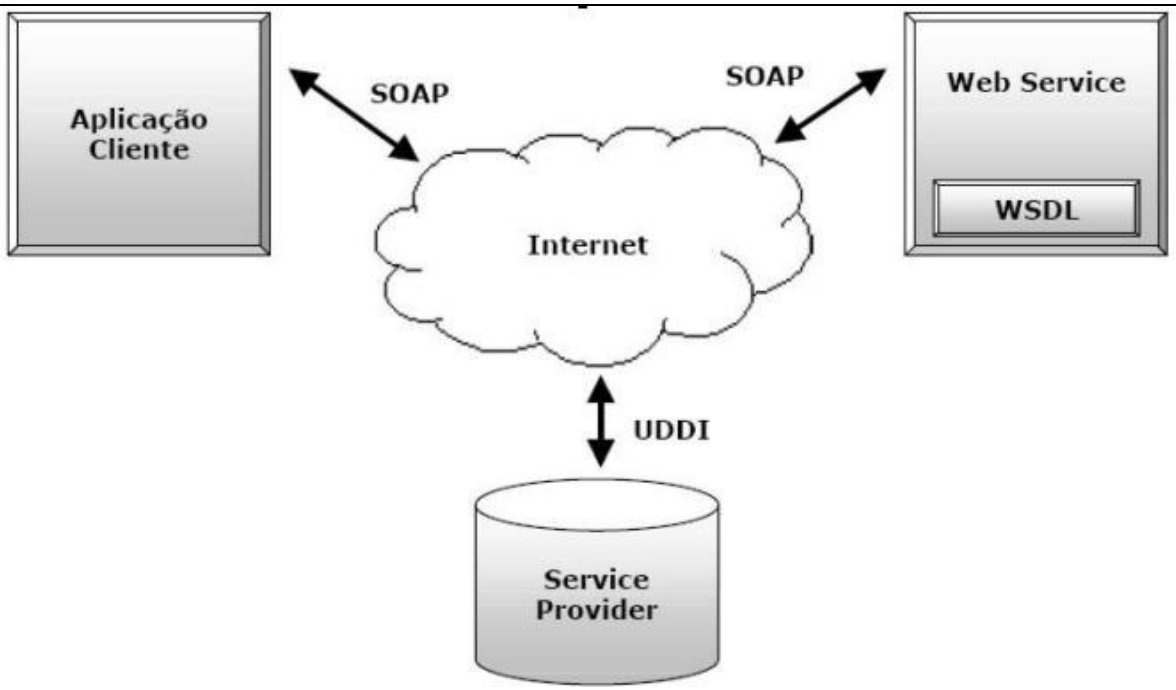

Figura 3 1 - Funcionamento básico de uma solicitação a um web Service

A arquitetura básica de um Web Service contém tecnologias capazes de trocar mensagens, descrever Web Services e publicar e descobrir aplicações Web Services.

O fundamento da arquitetura Web Service define uma interação entre agentes de software como um trocador de mensagens entre serviços requisitados e provedores de serviço.

Os requisitantes são softwares agentes que requisitam a execução do serviço. Provedores são agentes de software que provêem o serviço.

Provedores são responsáveis por publicar a descrição do serviço a ser disponibilizado, enquanto que, os requisitantes devem ser capazes de encontrar as descrições dos serviços. E os agentes podem ser ambos.

A interação envolve a publicação, localização e operação de união.

A Figura 3.2 ilustra a arquitetura básica do Web Service, na qual um requisitante de serviço e um provedor interagem baseados no serviço publicado pelo provedor.

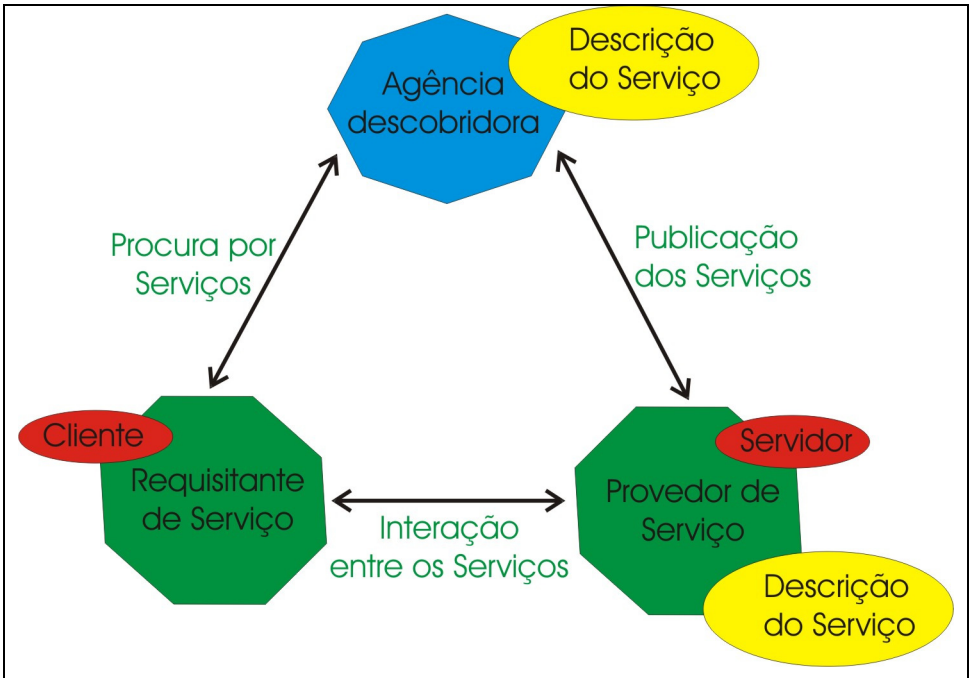

Figura 3 2 - Arquitetura de um web Service

A arquitetura básica de Web Service é tipicamente definida usando definições XML. A natureza da interface esconde detalhes da implementação do serviço, podendo ser usada independentemente de plataformas de hardware ou software.

A Figura 3.3 descreve as etapas para o acesso de um web Service e como interagem as tecnologias já citadas neste texto (UDDI, WSDL, SOAP).

Web Services podem ser usados sozinhos ou em conjunto com outros web Services.

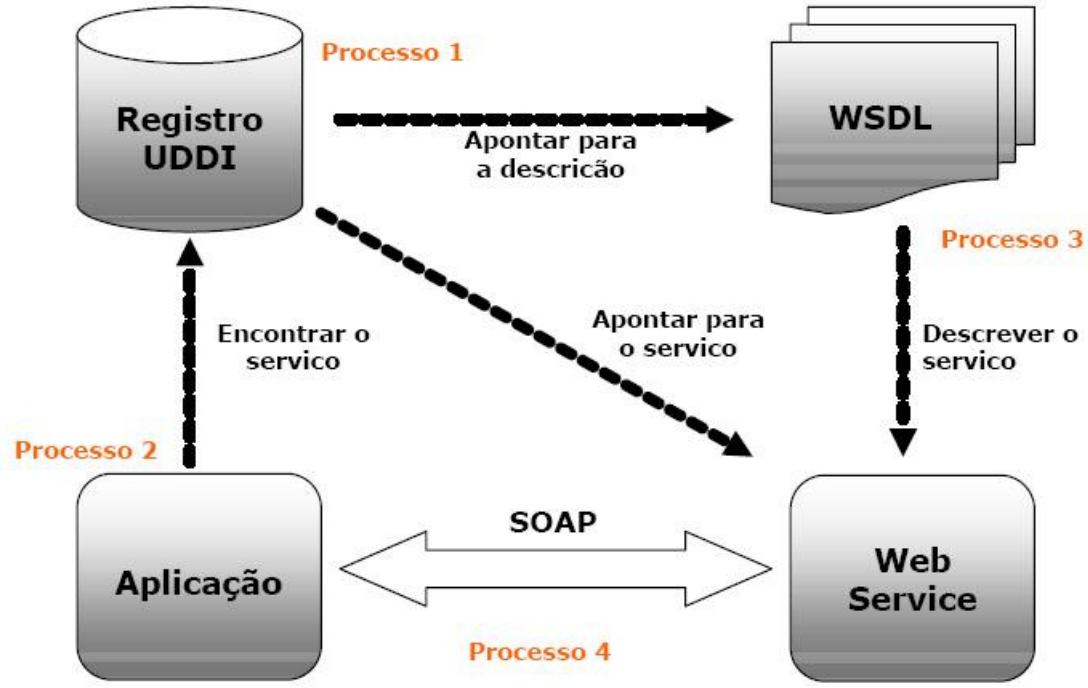

Figura 3 3 - Etapas de acesso a um web service.

No próximo capitulo são descritos as tecnologias para a construção do modelo MVC que é a base para o sistema construído, assim como, outras tecnologias que podem futuramente ser adicionadas ao modelo MVC.

Também são analisados dois modelos HMVC e MMVC para a construção de sistemas na web que buscam melhorar o modelo MVC.

CAPÍTULO 4- TECNOLOGIAS PARA O USO DE CONTEÚDO DINÂMICO NA WEB

Neste tópico serão apresentadas as tecnologias que servem de base para a construção do modelo MVC que são Servlets, JSP e beans, assim como o modelo MVC e suas principais características e vantagens, além de outras tecnologias e modelos que podem futuramente ser adicionados ao modelo MVC.

Também são destacados modelos que utilizam o MVC como base de sua construção como o HMVC e o MMVC.

Pode-se destacar alguns pontos em relação à construção de páginas web que as torna dinâmica como:

- Uma página web pode ser baseada em dados enviados pelo cliente.
- Uma página web pode ser derivada de dados que mudam freqüentemente e de acordo com a requisição de cada cliente.
- Uma página web pode usar informações de bancos de dados associados ou de outras fontes do lado do servidor.

As características descritas anteriormente desde que sejam necessárias para a implementação de uma página web diferenciam e explicam porque se deve utilizar páginas web dinâmicas em vez de páginas estáticas.

Uma das primeiras tecnologias para a construção de páginas web dinâmicas a surgir é *Common Gateway Interface* (CGI).

A CGI tem o funcionamento como mostrado na Figura 4.1, e é importante observar que a cada requisição HTTP realizada por um cliente ao servidor HTTP um novo processo é criado.

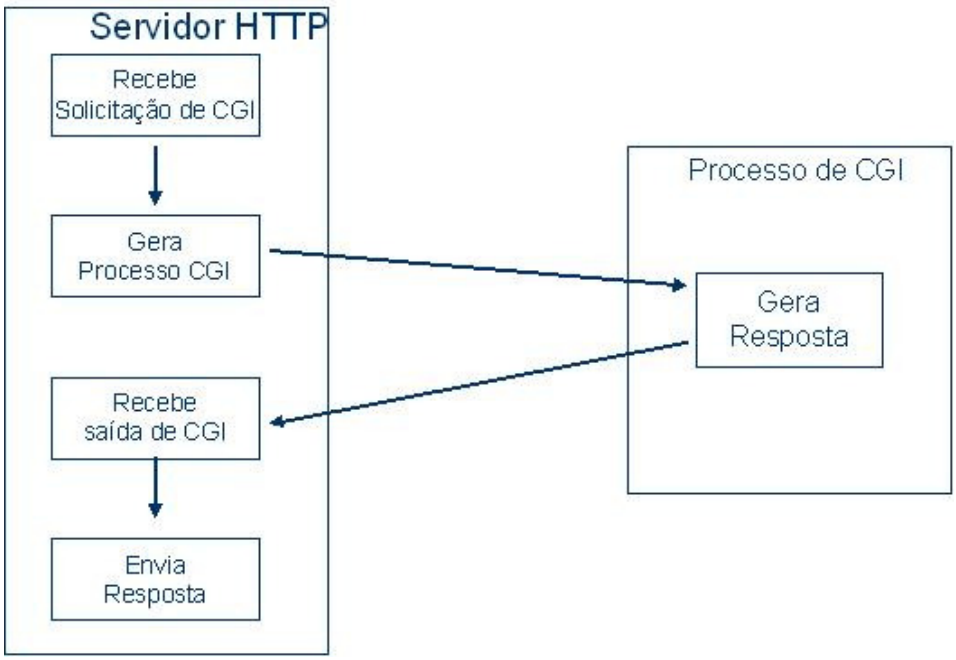

Figura 4 1 - Funcionamento da tecnologia CGI

São descritas ao decorrer deste capítulo varias formas de se gerar e desenvolver conteúdo dinâmico para a web, como: Servlets, JSP, JavaBeans e EJB.

4.1 A tecnologia dos Servlets

Da mesma forma que a Sun Microsystems criou as Applets como aplicações baseadas em Java com o objetivo de adicionar funcionalidades interativas a computadores clientes, a Sun também criou os Servlets, para serem aplicações também baseadas em Java, mas com o objetivo de adicionarem funcionalidades dinâmicas a servidores.

Servlets são programas Java que rodam em web ou servidores de aplicação, atuando como camada intermediaria entre requisições que chegam de navegadores web ou de outros clientes HTTP e bancos de dados e aplicativos no servidor HTTP (Hall & Brown, 2000).

A Figura 4.2 ilustra de maneira simples a requisição de um cliente (usuário final) a um Servlets, que por sua vez se faz de intermediário a outros aplicativos e em seguida retornando o resultado ao cliente.

As requisições podem ser feitas a vários serviços diferentes como podem ser vistas na Figura 4.2.

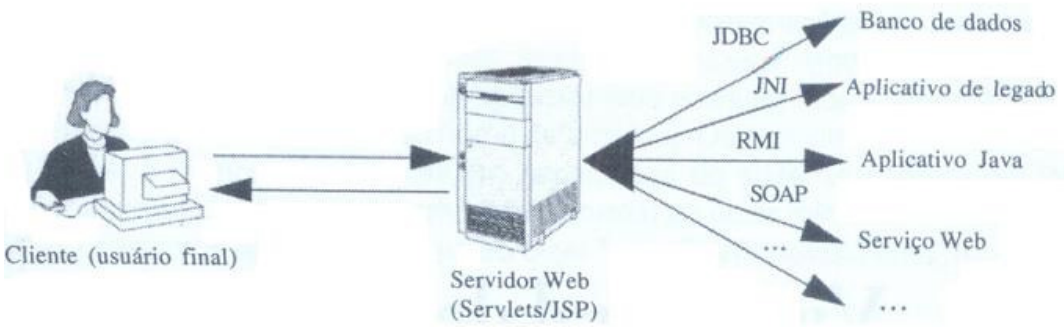

Figura 4 2 - O papel do middleware web

No decorrer de cada requisição os Servlets adquirem a responsabilidade de localizar e construir o conteúdo apropriado para a resposta do servidor.

A construção deste conteúdo segue os seguintes passos:

1- Ler o dado explícito enviado pelo cliente.

2- Ler o dado de requisição HTTP implícito enviado pelo navegador.

3- Gerar os resultados, que pode envolver uma comunicação com o banco de dados, uma chamada RMI ou EJB, invocar um serviço web ou retorna diretamente a resposta.

4- Enviar os dados Explícitos para o cliente.

5- Enviar os dados de resposta HTTP implícitos.

Dados explícitos são aqueles que são enviados por um formulário HTML comum, enquanto que, os dados implícitos são as informações HTTP que são enviadas, que podem ser cookies, tipo de mídia, outras.

Uma importante vantagem a ser observada na geração de conteúdo dinâmico pelos Servlets é a de ser escrito na linguagem Java, que permite que os Servlets se aproveitem de todos os benefícios da linguagem Java, como: a orientação a objetos, gerenciamento automático de memória, portabilidade compatível com várias plataformas e o acesso à ricas coleções de APIs de Java que possibilitam o acesso a banco de dados, servidores de diretório, recursos de rede e outros.

A tecnologia Servlets diferentemente da tecnologia CGI se beneficia por conseguir executar todas as suas solicitações dentro de um único processo, aumentado consideravelmente a eficiência do mesmo e a velocidade das respostas às solicitações.

Servlets são muito mais eficientes, mais fáceis de usar, mais poderosos, mais portáveis, mais seguros e mais baratos (menos custo/ tempo de construção) do que a CGI (HALL & BROWN, 2000).

Na Figura 4.3 é mostrado o funcionamento de um Servlets, desde o recebimento de uma requisição até o envio da resposta a quem solicitou o serviço.

A partir do momento em que o servidor HTTP recebe as solicitações, as mesmas são enviadas a um *container Servlets*, que é responsável por buscar as informações relacionadas à solicitação e encaminhá-las para a criação de encadeamento. Diferentemente da Figura 4.1 referente à tecnologia CGI, todas as solicitações ficam armazenadas em um único processo, permitindo um melhor desempenho do serviço e, automaticamente, uma resposta mais rápida.

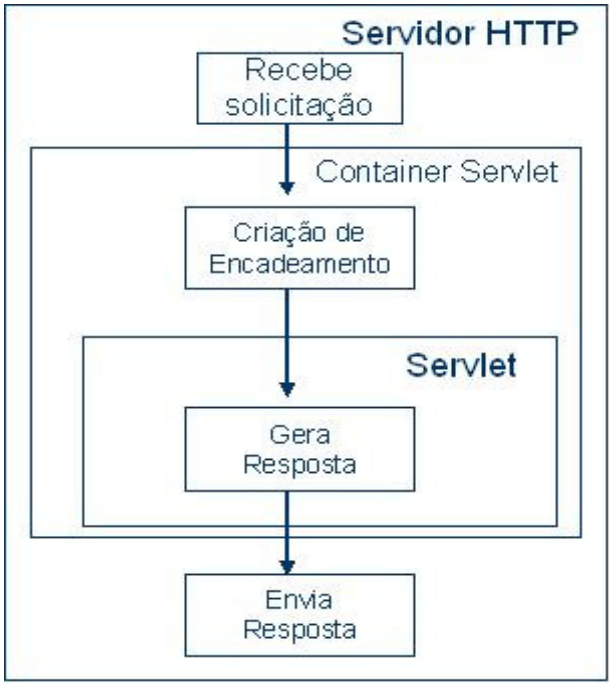

Figura 4 3 - Funcionamento de um Servlets.

Servlets são implementados através de um *Container Servlets* e associados com um servidor HTTP, logo, sempre que uma solicitação é feita ao servidor HTTP, o mesmo a encaminha para o *Container Servlets*, que por sua vez, encaminha a solicitação para uma instância da classe chamada que é empacotado de todos os dados da solicitação. Em seguida é criado um objeto Java, gerando assim, a resposta que depois é enviada para quem a solicitou.

A deficiência desta tecnologia é que todo o conteúdo dos documentos reside no código fonte do programa, qualquer modificação, seja de uma cor ou especificamente no layout da página precisaria de um programador, além da recompililação do Servlets.

A seguir é apresentado um exemplo básico de um Servlets, que produz uma página HTML para o cliente, imprimindo na tela um “Olá Mundo”, quando solicitado.

Observe que é um código Java comum, que importa as APIs: Java.io.*, javax.servlets.* e javax.servlets.http.*. A API java.io.* é necessária para o objeto

PrintWriter, a javax.servlet para o objeto HttpServlet e a javax.servlet.http para HttpServletRequest e HttpServletResponse, além de estender a uma classe padrão HttpServlet, que fornece uma infra-estrutura para lidar com HTTP.

O HttpServletsRequest permite sejam obtidos todos os dados de chegada, enquanto que, o HttpServletsResponse permite que sejam especificadas informações de saída.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Alô Mundo</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Pode-se observar que um Servlet gera o conteúdo de uma página HTML, que é apresentada como resposta a requisição do cliente, por isso, existe a dificuldade encontrada a cada vez que se precise modificar a cor de um texto ou simplesmente o tamanho do mesmo, é preciso recompilar a classe para este simples ato.

Visualização da página HTML gerada pelo Servlets quando este é requisitado.

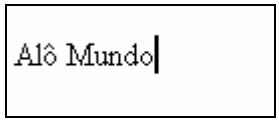

Figura 4 4 - Visualização da requisição de um Servlets.

Pode-se dizer que Servlets são programas Java com HTML embutido dentro deles e destaca-se para invocar a lógica de negócios e efetuar operações mais sofisticadas como tarefas orientadas para processamento.

Algumas desvantagens para o desenvolvimento de um sistema utilizando apenas a tecnologia Servlets.

- São Difíceis de escrever e manter o HTML.
- Não se pode utilizar ferramentas HTML padronizadas.
- O HTML é inacessível para desenvolvedores não Java.

No próximo tópico será apresentada a tecnologia JSP, que baseia-se na tecnologia Servlet, porém, apresenta-se de maneira diferente.

4.2 – A Tecnologia JavaServer Pages

Uma página JSP pode ser considerada uma página HTML comum com conteúdo JAVA embutido, quando uma página JSP é chamada, a mesma é traduzida em um Servlets que por sua vez é compilado, e em tempo de execução, são os Servlets compilados que são executados. Portanto, escrever JSP pode ser considerado como uma outra maneira de escrever Servlets.

Pode-se considerar algumas vantagens de JSP sobre Servlets.

- É mais fácil de escrever e manter o HTML.
- Pode-se utilizar ferramentas de desenvolvimento padronizadas de sites para a web.
- Pode-se dividir a equipe de desenvolvimento.

JSP se destaca por ser uma tecnologia destinada a duas categorias de programadores, *Web Designers*, que se preocupam mais com a aparência da página, ou

seja, a camada de apresentação, e a programadores, que escrevem os códigos Java que tratam do conteúdo dinâmico da página.

JSP é uma tecnologia baseada em Java que simplifica o processo de desenvolvimento de *sites* da web dinâmicos (FIELDS & KOLB, 2000).

JSP se diferencia de uma página HTML por permitir conteúdo dinâmico às páginas, ou seja, as informações estão em constantes mudanças, além da possibilidade de personalizar dados e interfaces para cada usuário. Essas são características importantes que uma página HTML, por ser estática, não possibilita aos usuários.

Uma página estática pode ser considerada aquela que não depende de dados ou informações a ela enviada para ser gerada ou atualizada, ou seja, independente de qual usuário a estiver acessando os dados e informações da página sempre serão os mesmos, a atualização do conteúdo da página pode ser feita periodicamente pelo desenvolvedor.

Já uma página dinâmica é atualizada ou age de acordo com o que o usuário a enviar, ou caso não envie nenhum dado ou informação, a mesma é atualizada com as opções padrão definidas pelo programador.

Um arquivo JSP é normalmente um arquivo de texto que tem a extensão ".jsp", e também *tags* JSP e HTML que não são obrigatórias. A página JSP quando solicitada a um servidor web retém toda a parte de código JSP e retorna ao usuário apenas o código HTML, isso faz com que toda a implementação da página fique em total segurança e principalmente oculta ao usuário final.

A seguir é apresentado um exemplo de código JSP que retorna um ao cliente a mensagem "Alô Mundo", caso não seja passado um valor para a variável nome, caso contrario, irá retornar o valor inserido na variável pelo usuário.

O exemplo é composto por uma página HTML que possui um campo chamado *nome.* O valor do campo nome é preenchido pelo usuário final que o envia para a página

JSP através do clique em um botão chamado enviar. O valor do campo nome é então enviado para a página hello.jsp.

Código da página HTML:

```
<html>
<body>
<form method="POST" action="/hello.jsp">
 Nome: <input type="text" name="nome" size="10">
 <input type="submit" value="enviar" name="B1">
</form>
</body>
</html>
```

Visualização do código HTML.

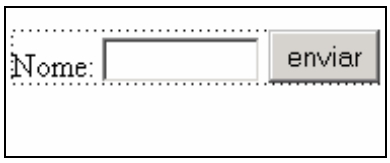

Figura 4 5 - Visulização de uma página HTML que faz requisição a uma página JSP.

Código da página hello.jsp que recebe o valor da variável nome enviado pela página HTML, após receber o valor exibe o mesmo.

```
<html>
<body>
<% String visitor = request.getParameter ("nome");
if (visitor == null) visitor = "Alô Mundo"; %>
<%= visitor %>
</body>
</html>
```

Visualização do código JSP que recebe o valor da variável nome.

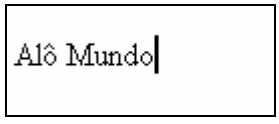

Figura 4 6 - Visualização da página JSP que recebe a requisição da página HTML.

No código JSP pode-se observar *tags* em HTML e *tags* em JSP trabalhando em conjunto. Nas *tags* JSP que nos interessam no momento é declarada uma string chamada *visitor* que recebe o valor da variável nome. O recebimento desta variável é feito utilizando-se o comando `request.getParameter`, caso o parâmetro *nome* não seja

passado, ou seja, tenha o valor null, a string *visitor* recebe o valor "Alô Mundo", imprimido assim a mensagem "Alô Mundo".

Por outro lado, se o variável nome tiver um valor, será impresso na tela o valor passado a variável *nome*.

Código retornado ao cliente quando o mesmo requisita o arquivo hello.jsp, passando o valor do parâmetro *nome* igual a " Alô João".

```
<HTML>
<BODY>
Alô João
</BODY>
<HTML>
```

Visualização do código quando a variável nome recebe o valor "Alô João".

Figura 4 7 - Visualização de um código dinâmico JSP.

O exemplo analisado mostra simplificadamente uma página dinâmica que pode ter o seu conteúdo modificado a cada solicitação, bastando a mesma ser requisitada com parâmetros diferentes, permitindo assim, a cada usuário personalizar a sua resposta.

Criar e manter o conteúdo dinâmico é difícil, por isso, luta-se para minimizar o custo de programação desses conteúdos. A Sun Microsystems propõe o JSP como uma tecnologia robusta e repleta de recursos Java, para a execução em um servidor.

Para a implementação do sistema proposto neste trabalhado há a necessidade da utilização de uma linguagem que permita o uso de um conteúdo dinâmico, pois, serão vários usuários ao mesmo tempo acessando as páginas dos sistemas e passando valores de variáveis diferentes, que retornarão respostas únicas para cada usuário final.

O JSP se encaixa perfeitamente nessas necessidades, além de ser baseado em Java, o mesmo, aproveita-se principalmente de vantagens desta linguagem como:

orientação a objeto, encapsulamento, tratamento de exceções e gerenciamento de memória automática, reuso entre outras.

É importante destacar que JSP é uma tecnologia que gera conteúdo dinâmico do lado servidor, diferentemente, por exemplo, do JavaScript que gera conteúdo dinâmico do lado cliente através do navegador.

É importante ressaltar que as JSP e JavaScript não são concorrentes e que até podem trabalhar em conjunto.

JSP tem como uma de suas principais vantagens à velocidade do processamento depois da primeira execução. A Figura 4.4 mostra o funcionamento de uma página JSP.

Na Figura 4.4 pode-se identificar que as solicitações do navegador da web são recebidas pelo servidor HTTP, sendo depois encaminhadas para o *Container* JSP. No *Container* JSP são feitas verificações como: primeira vez que a página JSP foi solicitada, ou se a página solicitada é mais atual do que a que está armazenada, caso verdadeiro uma das opções citadas, então é preciso que a página seja analisada, para então ser gerado o seu código fonte e depois compilá-lo. Só após estes passos é que a página pode ser carregada e enviada para o Servlet da página JSP que gera a resposta e depois a envia a quem a solicitou.

Este processo todo gera uma certa demora na resposta para uma solicitação, porém, isto tudo é feito apenas na primeira vez que uma página é solicitada.

Quando a solicitação é realizada pela segunda vez o processo de resposta torna-se mais rápido, principalmente se a página continua carregada no container JSP. É possível que uma página apesar de já ter sido carregada tenha que ser compilada novamente, caso a mesma não seja freqüentemente usada, sendo descarregada do

container JSP. Porém, se a página solicitada já estiver carregada o tempo de resposta de uma solicitação é notadamente minimizado, como se pode observar na Figura 4.8.

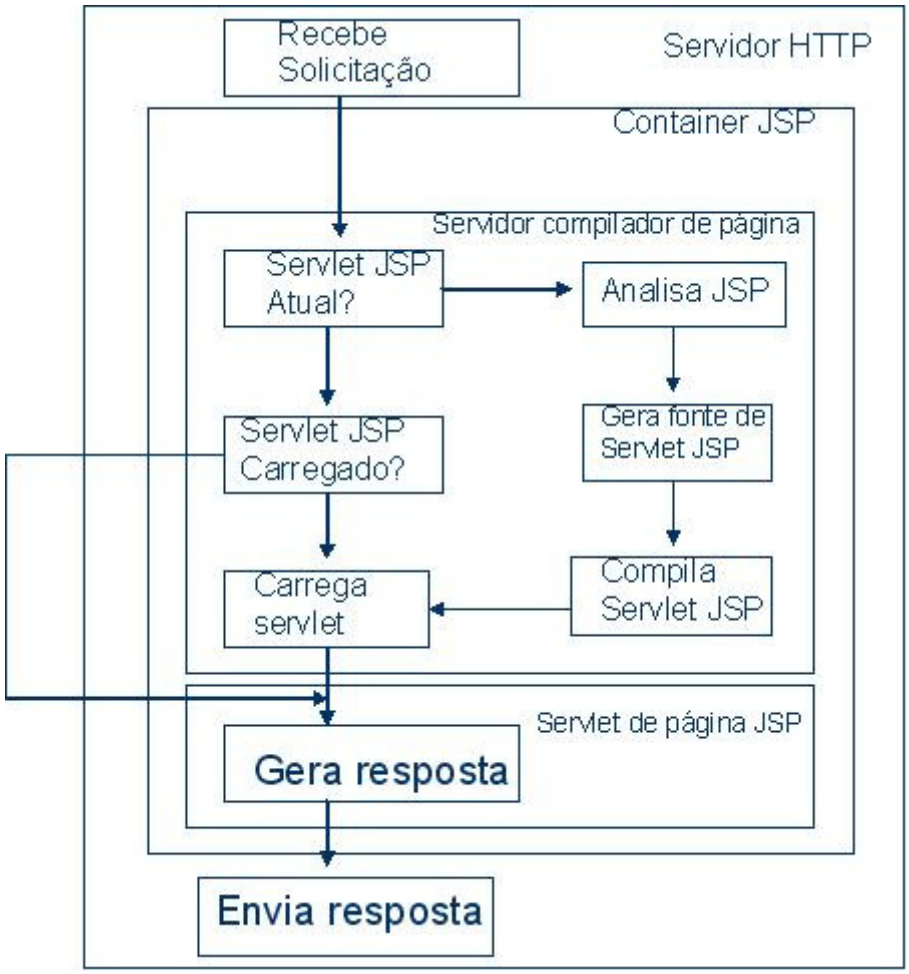

Figura 4 8 - Funcionamento de um JSP.

O JSP oferece vantagens como: separação da apresentação do conteúdo dinâmico, além de permitir a inserção de código Java na sua página, com as opções de scriplets, expressões JSP e diretivas. Para um melhor aproveitamento como será destacado nesse trabalho pode-se utilizar beans e tags personalizadas.

No próximo tópico será apresentado de uma maneira geral os Beans, que fazem parte da construção deste sistema proposto que tem como base o modelo MVC.

4.3 – Componentes Java Beans

São classes Java reutilizáveis que seguem algumas regras bem definidas para nomeação de seus métodos e variáveis. A idéia por trás do uso desses JavaBeans em páginas JSP, é que os mesmos encapsulem a lógica de uma aplicação, separando-a do restante da página.

Uma funcionalidade importante que vem classificando o JSP como uma das principais tecnologias no momento para desenvolvimento na web, é a possibilidade de separar a lógica da implementação da apresentação.

Uma das possibilidades de se conseguir esta separação é a implementação de um JavaBean, que nada mais é do que uma classe Java, com diversos métodos que podem ser acessados pelas páginas JSP.

A utilização de JavaBeans em conjunto com JSP possibilita a diminuição de código Java embutido nas páginas JSP, o que deixa uma página mais "limpa" e com menos dificuldade de entendimento para um webdesigner que não conhece a linguagem Java.

Uma outra observação a ser destacada é que a utilização de classes Java facilita na escrita, compilação, teste, depuração e principalmente no reuso das mesmas.

Beans são simplesmente classes Java padronizadas que tem algumas características como: não estendem nenhuma classe particular, não estão em nenhum pacote particular e não utilizam nenhuma interface particular (HALL & BROWN, 2000).

Algumas Características de um *bean* são listadas abaixo:

- Uma classe bean deve ter um construtor com nenhum argumento, ou não declarar o construtor, o que resulta em um construtor com nenhum argumentos criado automaticamente.

- Uma classe *bean* não deve ter campos de variáveis de instância pública.
- Valores persistentes devem ser acessados por métodos chamados getXxx e SetXxx.

Geralmente, objetos são criados e eliminados em função da execução do programa. Um objeto pode ser instanciado (criado) por um certo período de tempo e depois eliminado, liberando o espaço de memória ocupado, em um processo automático conhecido como coleta de lixo. É possível, porém, criar-se objetos persistentes, que continuam existindo mesmo depois do término do programa que os criou. Um exemplo de objetos persistentes seriam os armazenados em Banco de Dados Orientados a Objetos (WINBLAND, EDWARDS & KING, 1990).

Vantagens de se utilizar bean em uma página JSP sobre Scriptlets e expressões JSP.

- Nenhuma sintaxe Java
- Mais simplicidade no compartilhamento de objetos.
- Correspondência conveniente entre parâmetros de requisição e propriedades do objeto.

Portanto, qualquer modificação na implementação da classe Java, não interferirá em nada na apresentação e no layout da página, sendo que também, qualquer modificação na apresentação ou *layout* da página não interfere na implementação do código Java que é utilizado.

A seguir é mostrado um exemplo de código de uma página JSP acessando um JavaBean, através de *tags* especificas para acesso aos JavaBean.

No código se percebe o uso do HTML e das *tags* JSP que servem para o acesso de um JavaBean, a primeira *tag* JSP, tem como finalidade dizer o acesso a um bean “jsp:useBean”, nesta *tag* são declarados um identificador id=”hello” que servirá para

identificar o Bean a ser chamado pela página JSP e a opção class=”bean.HelloBean” que identifica a classe Java a ser utilizada como Bean, como ilustra o exemplo é necessário inserir o caminho completo do Bean, ou seja, o pacote em que está localizada a classe Java.

Na segunda *tag* JSP “`jsp:setProperty`” são declaradas algumas propriedades como nome do Bean a ser usado name=”hello”, a seguir é declarado a variável que vai ser alterada no Bean property=”nome”, ou seja, qual o campo do Bean que será modificado, e em seguida o valor para esta variável value=”Mundo”.

A Terceira *tag* JSP “`jsp:getProperty`” é para recuperar as informações solicitadas ao Bean, declara-se novamente o nome do identificador do Bean a ser acessado name=”hello” e depois a variável que se deseja recuperar property=”nome”.

```
<HTML>
<BODY>
<jsp:useBean id="hello" class="bean.HelloBean"/>
<jsp:setProperty name="hello" property="nome" value="Mundo"/>
Alô, <jsp:getProperty name="hello" property="nome"/>
</BODY>
</HTML>
```

Visualização da página quando a mesma é chamada.

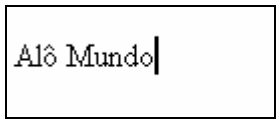

Figura 4 9 - Visualização de um Bean

Código do Bean HelloBean que é acessado pelo exemplo JSP .

```
package bean;
public class HelloBean
{
 private String nome;
     public String getNome() {
      return nome;
     }
    public void setNome(String n) {
     nome=n;
    }
}
```

No código da implementação do *Bean* se percebe que é uma classe Java normal, com métodos declarados. Esta classe chama-se HelloBean e está no pacote bean. Como a classe não possui variáveis de instância publicas e um construtor com zero argumentos, já que o mesmo não é declarado, além dos métodos de alteração "setNome" e recuperação "getNome" pode-se afirmar que satisfaz aos critérios básicos para ser um bean.

Uma característica importante a ser destacada nos *Bean* é o possível compartilhamento dos mesmos, através de uma opção nas páginas JSP chamada *scope*, que pode ter as seguintes opções: *page*, *request*, *session* e *application*.

Quando utilizamos o *scope*, o *bean* passa a ter um comportamento diferente dependendo da opção do mesmo, o sistema procura primeiro um *bean* existente de nome especifico na localização especificada, caso não encontre então um *bean* é criado, caso contrario, utilizará o encontrado.

São citadas as opções de *scope* para páginas JSP quando se trabalha com JavaBeans.

- scope="page": Este é o valor padrão, o mesmo comportamento é obtido caso seja omitido a opção *scope*. Esta opção indica que o bean não é compartilhado e desse modo um novo bean será criado a cada requisição.
- Scope="request": nesta opção é possível o compartilhamento entre páginas JSP ou entre páginas JSP e Servlets.
- Scope="session": nesta opção os beans são compartilhados para um único usuário enquanto a sessão for verdadeira, ou seja, enquanto o usuário estiver dentro do sistema.

- Scope=”application”: esta opção disponibiliza o bean através de uma variável *application* armazenada no servletContext, sendo este compartilhado por todos os Servlets e páginas JSP no aplicativo web. Nesta opção após a modificação de um bean este passa a ser visto modificado não apenas pelo usuário que o modificou.

Esta abordagem de compartilhamento de beans é essencial para a construção do sistema proposto por este trabalho.

4.4 - O Modelo MVC (Modelo Visualização Controlador)

O Modelo MVC surge da necessidade de páginas JSP, Servlets e Beans em se completarem para explorar o que cada um tem de melhor. Servlets são melhores para processamento de dados como: ler e checar dados, se comunicar com bancos de dados, chamar e criar a lógica de negócios (beans), enquanto que, páginas JSP são melhores na apresentação, que explora a construção do HTML para representar os dados de requisição e as *tags* especificas JSP.

O Modelo MVC é uma arquitetura de software que separa a parte de apresentação da aplicação de sua parte de implementação, dividindo a sua estrutura em três partes, de forma que cada parte tenha o mínimo de influência sobre a outra: modelo, visão e controlador.

O Modelo é representado por beans.

A Visão é representada por páginas JSP.

O Controlador é representado por Servlets.

4.4.1 – Introdução ao modelo MVC

O Modelo MVC surge a partir da implementação da interface de usuário Smalltalk-80 (BURBECK, 1992).

Recentemente este modelo tem inspirado no desenvolvimento de uma grande quantidade de sistemas para a Internet baseando-se nos modelos MVC existentes, possibilitando uma melhor divisão em grandes projetos para o desenvolvimento de sistemas grandes e complexos e também apresentando uma alternativa para modelos pequenos e simples, mais a frente serão apresentados os modelos MVC 1 e MVC 2.

O desenvolvimento de uma aplicação para a web utilizando este modelo passa a ser dividida em 3 partes: modelo, visão e controlador. Estas três partes dividem-se em apresentação e implementação como já citado. A apresentação preocupa-se apenas com a parte de visualização da página, ou seja, o seu layout, que passa a ser desenvolvido por um especialista na área, um webdesigner, enquanto que, a implementação deve oferecer todas as respostas necessárias a ações que venham a ocorrer na mesma, esta parte do projeto deve ser de responsabilidade de programadores, que preocupa-se com toda a parte de desenvolvimento da aplicação.

Com esta divisão na construção de uma aplicação para a *web*, seguindo este modelo tanto o *webdesigner* como o programador necessita apenas do conhecimento especifico de sua área e o mínimo necessário da área do outro.

Uma das finalidades da construção de um ambiente na arquitetura MVC é aproximar cada vez mais uma aplicação *desktop* de uma aplicação para a Internet. De maneira que uma aplicação para a *web* se torne num futuro bem próximo uma aplicação *desktop*.

É possível que grande parte das aplicações de hoje passem a ser desenvolvida para a web, de maneira que aplicações *desktop*, necessitem se adaptar a este contexto ou já "nascerem" para a web.

A arquitetura do modelo MVC possibilita e será utilizada neste trabalho para oferecer uma maior organização no desenvolvimento de aplicações para a web.

As tecnologias utilizadas para a construção do modelo MVC são JavaServer Pages, Servlets e JavaBeans. Como já apresentado anteriormente é ilustrado o modelo MVC na Figura 4.10 e os suas respectivas partes: modelo, visualização e controle.

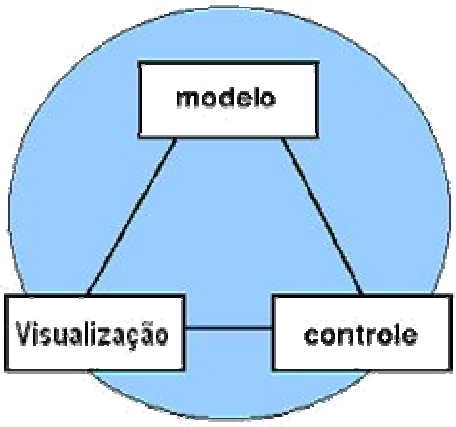

Figura 4 10 - Modelo MVC.

4.4.2 - Modelo MVC 1

Este modelo define-se por ser um modelo formado apenas por páginas JSP e Beans. É um modelo que deve ser utilizado para o desenvolvimento de sistemas pequenos e simples, no qual pode haver uma grande quantidade de código Java nas páginas JSP.

A Figura 4.11 ilustra o modelo MVC 1.

A seguir é definido o funcionamento do modelo MVC 1.

1. É enviada uma requisição à página JSP.

2. A página JSP que foi solicitada cria e chama o bean para passar a requisição.
3. O bean faz conexão com o banco de dados e grava as informações.
4. A página JSP responde ao navegador (browser) do usuário enviando a resposta à requisição.

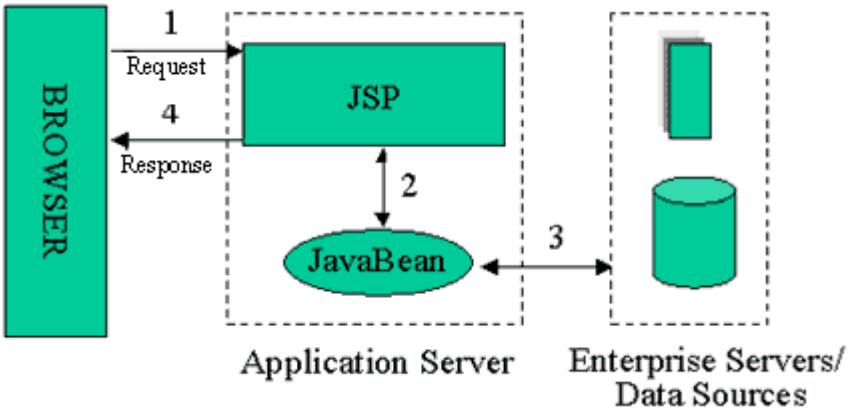

Figura 4 11- Funcionamento do modelo MVC 1.

4.4.3 - Modelo MVC 2

Este modelo MVC 2 é o modelo que serve de base para a construção do sistema proposto por este trabalho. Este modelo explora o que há de melhor em cada tecnologia, dividindo o desenvolvimento do sistema em duas partes: implementação e apresentação, além de adicionar um componente a mais em relação ao modelo MVC 1, os Servlets.

A Figura 4.12 mostra o funcionamento da arquitetura modelo MVC 2.

A seguir é apresentado o funcionamento do modelo MVC 2.

1. O Controlador recebe a requisição feita pelo navegador do usuário.
2. O Controlador Cria o bean e repassa a requisição ao bean.
3. O Controlador escolhe a qual página JSP deve enviar a requisição.

4. A Página JSP acessa o bean para recuperar a requisição solicitada, caso esteja no banco de dados este é acessado também.
5. A resposta é visualizada pelo navegador do usuário que fez a requisição.

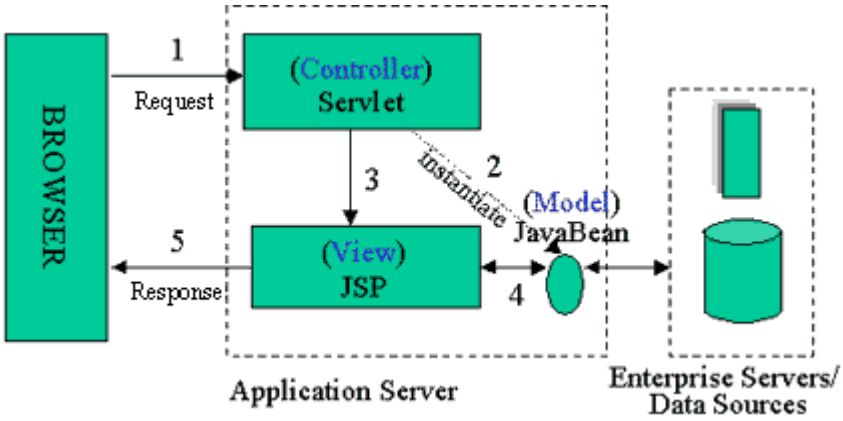

Figura 4 12 - Funcionamento do modelo MVC 2.

Pode-se destacar alguns passos para a construção do modelo MVC 2 como:

1- Definir beans para representar os dados.

2- Usar um Servlets para tratar requisições.

3- Preencher os beans.

4- Armazenar o bean na requisição, sessão, ou no contexto do Servlets.

5- Encaminhar a requisição para uma página JSP.

6-Extrair os dados dos beans.

Com esta divisão fica mais fácil o controle de tarefas e ação feitas pela a implementação do sistema, assim como, a parte de apresentação da parte de visualização.

4.5 – Tags personalizadas

Uma outra maneira de se utilizar o modelo MVC é introduzir elementos de referência a uma biblioteca de *Tags* em páginas JSP. Sendo que este modelo se baseia no modelo MVC.

A Figura 4.13 ilustra a separação entre apresentação e implementação de um sistema para a web utilizando tags personalizadas.

Figura 4 13 - Tags personalizadas

Uma diferença entre acessar Beans através de páginas JSP e acessar uma biblioteca de *Tags* através de páginas JSP é que esta oferece acesso nativo aos objetos das páginas JSP.

Uma biblioteca de *Tag* permite a separação da lógica de programação de sua aplicação do conteúdo da pagina JSP, assim como o javaBeans.

A seguir são apresentados os componentes necessários para a construção de uma biblioteca de tag:

1- Descritor de bibliotecas de tags (*Tag Library Descriptor*): este arquivo deve ser de extensão ".tld" e colocado no diretório WEB-INF da aplicação, além de conter as configurações das bibliotecas de Tags utilizadas na aplicação.

2- *Deployment Descriptor*, que é o arquivo chamado "web.xml", que fica localizado dentro do diretório WEB-INF, no qual contem informações de configuração da aplicação, como por exemplo: parâmetros de inicialização, mapeamentos de Servlets e neste caso, mapeamento de uma URI, referenciada na página JSP a biblioteca de Tags.

3- *Tag Handler*: classe que gerencia a *Tag*, que é referenciada pelo Descritor da biblioteca de Tags pelo elemento tagclass, além de ser responsável por executar as rotinas pertinentes quando a *Tag* é encontrada na página JSP.

4- A pagina JSP que chama a biblioteca.

A seguir um exemplo de implementação de uma página JSP que acessa uma biblioteca de Tags e todos os arquivos necessários para o funcionamento.

Este exemplo exibe um texto com a mensagem "Alô Mundo".

- Código do descritor de bibliotecas de *Tags*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library 1.1//EN"
 "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
 <taglib>
     <description>Exemplo de Biblioteca de tags</description>
     <tlibversion>1.0</tlibversion>
     <jspversion>1.2</jspversion>
     <short-name>Exemplo Tags</short-name>
     <uri>alomundo</uri>
     <tag>
         <name>alomundo</name>
         <tag-class>tld.AloMundo</tag-class>
        <body-content>empty</body-content>
     </tag>

 </taglib>
```

- Código do Deployment Descriptor

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
     xsi:schemaLocation=http://java.sun.com/xml/ns/j2ee
     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd
     version="2.4">

<display-name>TagLib</display-name>

<welcome-file-list>
     <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

<taglib>
     <taglib-uri>exemplo</taglib-uri>
     <taglib-location>/WEB-INF/tld/exemplo.tld</taglib-location>
</taglib>

</web-app>
```

- Tag Handler: código da classe que controla a biblioteca de Tags.

```
package tld;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import java.io.IOException;

public class AloMundo extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
     getJspContext().getOut().write( "Alô Mundo" );
    }
}
```

- Página JSP que acessa a biblioteca de Tags.

```
<%@ taglib uri="alomundo" prefix="tag" %>
<html>
  <body>
   <tag:alomundo/>
  </body>
</html>
```

- Visualização da página JSP

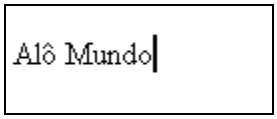

Figura 4 14 - Visualização de uma página JSP que consulta uma biblioteca de Tags

4.6- Frameworks que utilizam o modelo MVC

Dê uma forma geral um framework é um conjunto de classes e interfaces que cooperam para resolver um tipo de problema de software. Estes problemas podem ser dificuldade em desenvolver alguma ação ou simplesmente o desenvolvimento de um projeto mais complexo, que necessite de uma melhor organização e estrutura.

De acordo com a definição de Grady Booch: "*Um framework pode ser visto como um padrão de arquitetura cuja modelagem reflete uma infra-estrutura reutilizável e adaptável a algum contexto".*

Um bom framework deve oferecer um comportamento genérico que diversas aplicações façam uso.

Um framework, em geral, apresenta as seguintes características:

- É composto por múltiplas classes ou componentes, cada um devendo prover uma abstração de um conceito particular;
- Define como essas abstrações trabalham juntas para resolver um problema;
- Seus componentes são reusáveis;
- Organiza padrões em alto nível.

Existem muitas interpretações do que o framework é constituído. Alguns consideram as classes e interfaces fornecidas pela plataforma Java(JDK) um framework.

Na realidade, esse conjunto de classes constitui uma biblioteca. É importante saber a diferença entre uma biblioteca e um framework. Uma biblioteca contém funções e rotinas que são invocadas por uma aplicação, enquanto que, um framework provê componentes cooperativos e genéricos que a aplicação herda para fornecer um conjunto particular de funções.

Um framework é uma extensão de uma linguagem mediante uma ou mais hierarquias de classes que implementam uma funcionalidade e que (opcionalmente) podem ser estendidas. O framework pode envolver bibliotecas de tags.

Existem vários frameworks que auxiliam no desenvolvimento de aplicações web. Serão citados alguns dos principais frameworks que oferecem o desenvolvimento para aplicações WEB e apresentado o framework Struts e o JSF que implementam a arquitetura do modelo MVC.

Alguns tipos de frameworks:

- Barracuda
- Cocoon
- Expresso
- Freemaker
- Velocity
- webMacro
- Maverick
- SiteMesh
- Turbine
- Struts
- JavaServer Faces

4.6.1-Struts

A Struts Framework é um projeto *open source* mantido pela Apache Software Foundation. Foi escrita por Craig McClanahan em Maio de 2000, e desde então vem sendo melhorado pela comunidade *open source*. Tem como objetivo fornecer uma framework baseado no modelo MVC 2 para facilitar o desenvolvimento de aplicações para web.

A arquitetura MVC como já citado é um padrão que define a separação de maneira independente do *Model* (Modelo) que são os Objetos de Negócio, da *View*

(Visão) que compreende a interface com o usuário ou outro sistema e o *Controller* (Controlador) que controla o fluxo da aplicação.

- Como funciona o Struts em aplicações web?

O navegador gera uma solicitação que é atendida pelo Controlador (um Servlet especializado), e o mesmo se encarrega de analisar a solicitação e, segue a configuração que está programada em seu arquivo XML e chama a *Action* correspondente passando lhe os parâmetros enviados. A *Action* instanciará os objetos de negócio para completar a tarefa. De acordo com o resultado que a *Action* retornar, o Controlador irá direcionar a geração de interface para uma ou mais páginas JSP, as quais poderão consultar os objetos do Modelo a fim de realizar a sua tarefa.

A Figura 4.15 detalha o funcionamento do framework Struts.

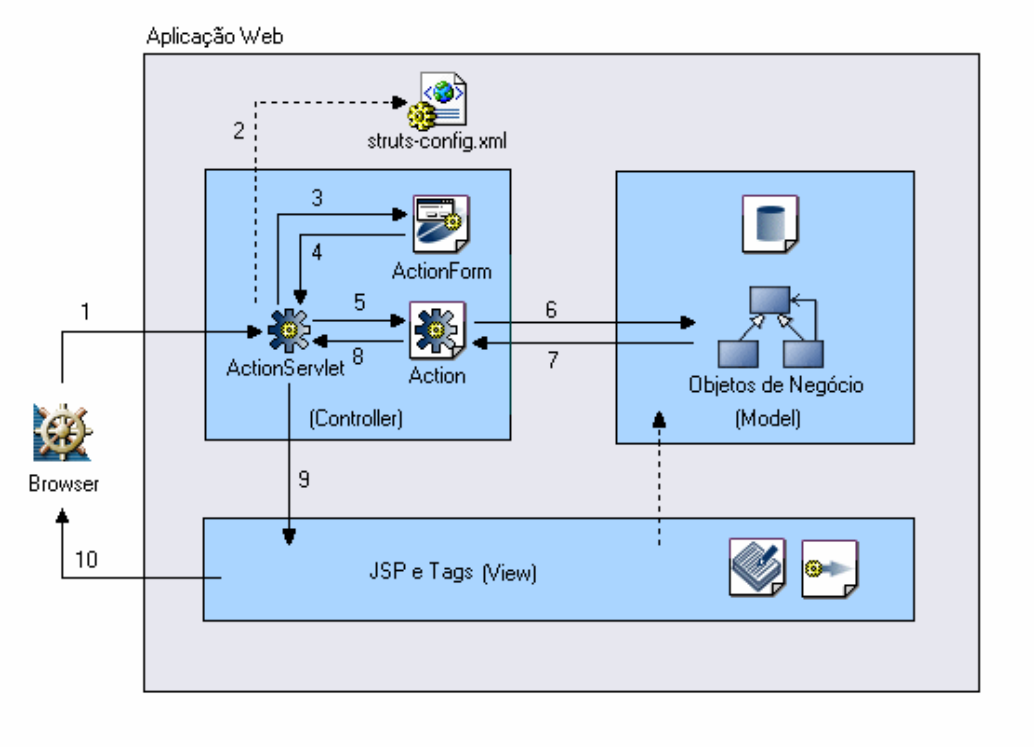

Figura 4 15 -Funcionamento do framework Struts

Segue a descrição dos passos realizados desde a requisição feita pelo navegador do cliente até a resposta recebida pelo mesmo.

1. O usuário faz uma requisição.

2. Se for a primeira solicitação que o container recebeu para esta aplicação, ele irá invocar o metodo init() da ActionServlet (controlador da Struts) e irá carregar as configurações do arquivo struts-config.xml em estruturas de dados na memória. Baseado no fluxo definido no arquivo struts-config.xml, o ActionSerlet identificará qual o ActionForm (classe para a validação dos dados) irá invocar.

3. O controle da aplicação é retomado para a ActionServlet, que verifica o resultado da verificação do ActionForm.

4. O ActionServlet, baseado no fluxo da aplicação (estruturas já carregadas em memória) invoca uma classe Action. A classe Action passará pelo método *execute* que irá delegar a requisição para a camada de negócio.

5. A camada de negócio irá executar algum processo (geralmente um bean). O resultado da execução deste processo será usado na camada de apresentação para exibir os dados.

6. Quando o controle do fluxo da aplicação voltar ao Action que invocou o processo da camada de negócio, o resultado é analisado e defini-se qual o mapa adotado para o fluxo da aplicação.

7. Baseado no mapeamento feito pelo o Action, o Controller envia a uma página JSP para apresentar os dados.

8. Na camada de apresentação (View), os objetos que foram setados como atributos da sessão do usuário serão consultados para montar o HTML para o navegador.

9. Ao usuário final é apresentado apenas o HTML da página de reposta.

A geração da interface é feita através de tags personalizadas, também já implementadas pela Struts, evitando assim o uso de Scriptlets, deixando o código JSP mais limpo e fácil de manter.

4.7– HMVC

Apresenta-se como um Modelo que apresenta significamente a diminuição de riscos e custos no desenvolvimento de uma arquitetura Java para o cliente. O HMVC é aplicativo tanto para aplicações Java como para Applets, além de qualquer outra linguagem orientada a objetos que queira usufruir desta arquitetura.

O HMVC baseia-se no modelo MVC 2 para a sua construção.

Segundo esta Arquitetura que foi desenvolvida para superar alguns problemas encontrados com o modelo MVC em relação a GUI, como: não trata a complexidade do gerenciamento de dados, gerenciamento de eventos e fluxo de aplicações.

Logo uma adaptação ao MVC faz surgi o *Hierarchical Model View Controller* (HMVC).

HMVC tem como proposta oferecer uma robusta e um fácil entendimento da metodologia de *design* da camada para o desenvolvimento de uma completa camada de apresentação (Usuário). Enquanto o MVC oferece uma *framework* eficiente para o desenvolvimento de interações GUI, O HMVC adapta isto por inteiro para o cliente.

Alguns benefícios desta arquitetura são:

- Define a comunicação entre as camadas e isola de camadas mais altas.
- Define a comunicação com as camadas de fora com o mínimo acoplamento. (unir)
- Localização da exposição (direção) para o código da terceira parte.

O Padrão HMVC baseia-se na arquitetura MVC e decompõem-se da hierarquia de pais e filhos como mostra a Figura 4.16.

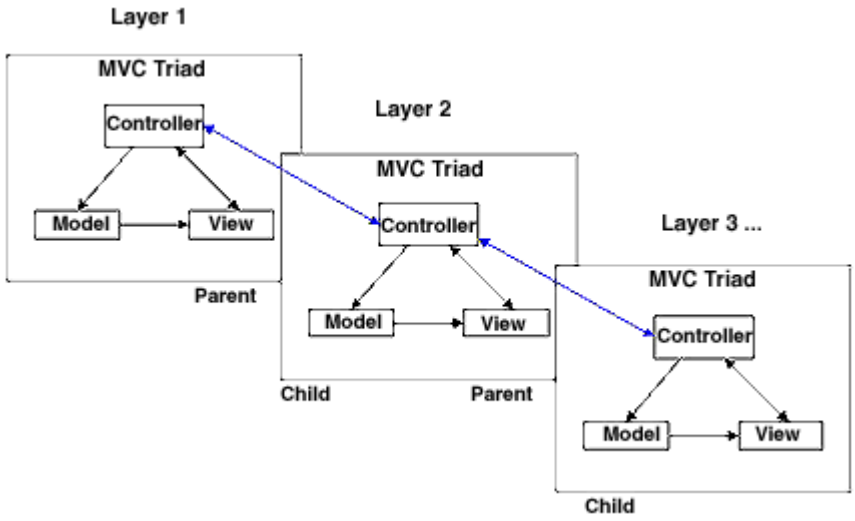

Figura 4 16 -Arquitetura da camada HMVC.

Alguns benefícios que a arquitetura HMVC revela os benefícios da orientação a objetos como: Redução da dependência entre partes muito diferentes do programa, reuso do código, componentes e módulos, além de extensibilidade aumentada facilitando a sustentabilidade.

A Figura 4.17 ilustra as camadas e componentes do modelo HMVC. Na horizontal é especificada a hierarquia entre a aplicação e na vertical os componentes do MVC. Entre as camadas o Controller tem a total responsabilidade de gerenciar os modelos e os componentes de visualização.

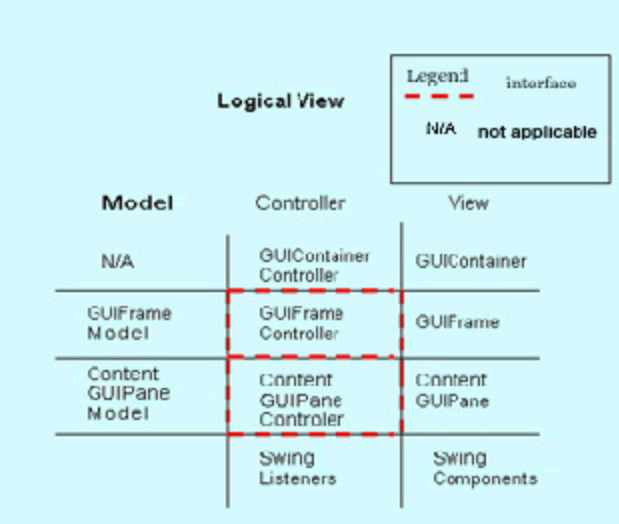

Figura 4 17- Camada HMVC e componentes.

São citados três aspectos para o desenvolvimento da camada de Apresentação (Usuário).

- Código da camada GUI:
- Código da característica GUI: Validação e captura de eventos do usuário.
- Código lógico da aplicação: fluxo, navegação, interação com o servidor.

O HMVC caracteriza-se por ser um modelo para melhorar as aplicações do lado cliente.

4.8- MMVC (*message-based* MVC)

O Objetivo da dissertação de (QIU, 2005) é desenvolver um paradigma para a próxima geração de aplicações software com uma arquitetura "limpa" que unifica (aproxima) aplicações *desktop* e aplicações web.

Por isso (QIU, 2005) apresenta como proposta o modelo MMVC que tem como centro o desenvolvimento para aplicações distribuídas.

O modelo de arquitetura de software MMVC busca aproximar aplicações web, aplicações distribuídas e aplicações colaborativas na Internet. Esta aproximação permite um maior reuso dos componentes existentes, esquema mais flexível com alta

escalabilidade e uma colaboração automática e eficiente com interatividade com um rico conteúdo de mídia para diversos clientes sobre ambientes de redes heterogêneas.

Além de uma sugestiva aproximação a uma interface uniforme para a próxima geração de clientes web com acessibilidade em todos os lugares. Esta arquitetura MMVC foi aplicada diretamente a um exemplo complexo de um browser SVG e deu a dimensão detalhada da performance para demonstrar a viabilidade desta aproximação.

O Modelo MMVC considera o modelo e a visualização como componentes distribuídos de uma aplicação que é conectada por mensagens. Neste sistema não existe apenas uma classe controladora.

Na Figura 4.18, os eventos UI (interface de usuário) formam a componente view (visualização), os eventos UI de nível mais alto e eventos semânticos incluídos formam o model (modelo) e os eventos de mensagem que tem a função de ligar o modelo e a visualização são o Controlador.

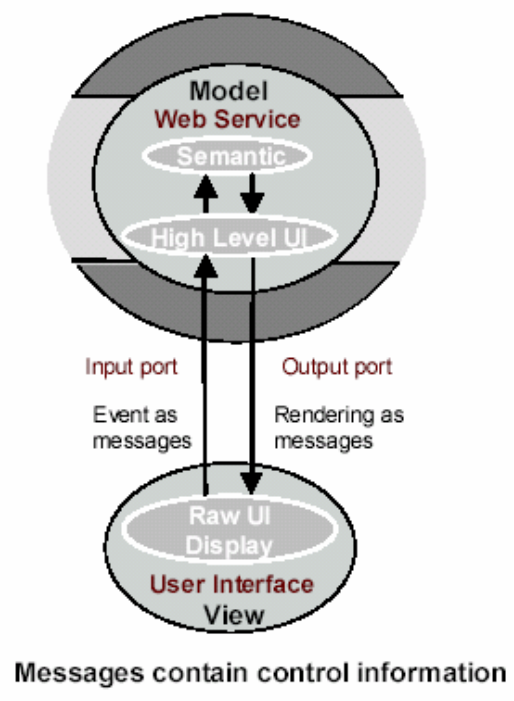

Figura 4 18 - Modelo MMVC.

O MMVC além de se basear no modelo MVC 2 para a sua construção também baseia-se em dois outros modelos *Multiple Model Multiple View* (MMMV) e *Single Model Multiple View* (SMMV) são arquiteturas gerais de colaboração baseada em mensagens com o modelo Web Service.

A Figura 4.19 mostra como o SMMV e o MMMV respectivamente são desenvolvidos com a arquitetura MMVC, são representados os modelos e as visuslizações, assim como os controles que são representados através de serviços de mensagens NaraBroking (QIU, 2005).

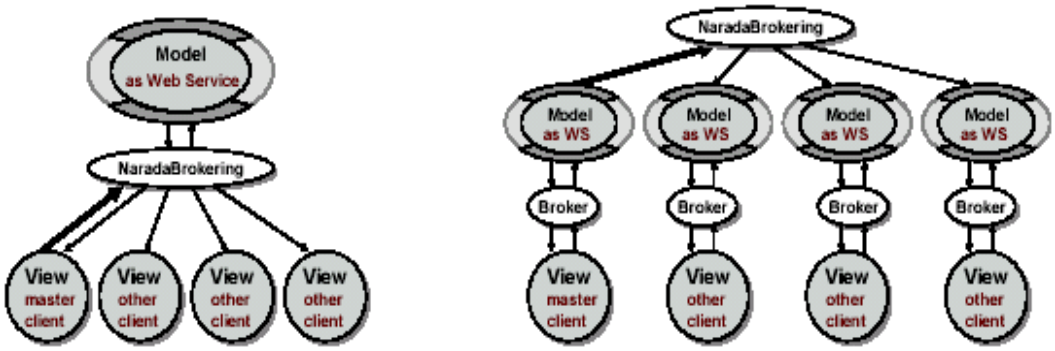

Figura 4 19 - Modelo SMMV e MMMV.

No próximo capitulo é descrito o SVG, linguagem gráfica baseada no XML que faz parte do desenvolvimento do sistema proposto, especificamente como a parte de visualização do sistema.

CAPÍTULO 5 – O AMBIENTE GRÁFICO SVG

O sistema proposto neste trabalho utiliza o formato SVG que é uma linguagem para a construção de gráficos em duas dimensões e também para a descrição de aplicações gráficas em XML, pode-se, porém estender este conceito para todo uma aplicação gráfica. Além dos conceitos e características gerais apresentados, também serão descritas as suas principais áreas de aplicação, assim como, características gráficas e dinâmicas do formato SVG. Alguns exemplos básicos de código também estão construídos para este propósito.

São áreas de aplicação: computação móvel, impressão digital, aplicações para web e intercâmbio de dados (GIS, mapas, área médica, etc.).

Pode-se considerar que a aplicação do formato SVG é a representação gráfica de algo. No entanto, desta representação podem surgir diversas áreas se aplicação como:

- Desenhos e Animações em páginas web;
- Exemplos interativos educativos;
- Slides;
- Mapas geográficos;
- Representação gráfica de dados estatísticos;
- Impressão de alta resolução

O formato e um ambiente do SVG são descritos para a geração de objetos e gráficos no formato SVG, baseado no modelo de desenvolvimento para sistemas na web, MVC.

5.1- Características gerais do formato SVG

Objetos no formato SVG são imagens vetoriais que apresentam significado semântico. Este significado semântico é definido na construção do objeto e é a representação gráfica da informação.

A semântica da imagem SVG permite que a mesma seja definida com um significado, ou seja, para a imagem de um círculo, o mesmo pode ter seu conteúdo o nome círculo. Na procura ou consulta por objetos definidos com o nome círculo em uma página da web, por exemplo, a imagem com o conteúdo de descrição do circulo é encontrada.

A característica vetorial permite que uma imagem seja redimensionada sem a perda da qualidade de visualização da imagem no momento em que a mesma é exibida, por exemplo, para se desenhar um objeto círculo, se define o valor de seu raio e as coordenadas x,y (pontos que determinam o centro do circulo) dentro de uma *tag* SVG definida <circle></circle>, sendo assim, sempre que haja a requisição do aumento da visualização do objeto, o mesmo é redesenhado na tela aumentando-se o raio e atualizando os valores de suas coordenadas (x,y).

Grande parte dos gráficos encontrados hoje na Internet se caracterizam por serem formadas por *pixels* (*picture element*), que se define como a menor parte de uma imagem digitalizada. Um *pixel* possui a informação que determina sua cor, sendo assim, para a formação de uma imagem, um conjunto de *pixels* é organizado no formato de uma matriz. O *pixel* determina a resolução desta imagem.

Uma característica importante de imagens formadas por *pixels* é a excelente visualização dos mesmos que depende de sua resolução, quanto maior a resolução melhor será definida e detalhada a imagem, porém, maior será o tamanho do arquivo.

Sendo objetivo deste trabalho a visualização de imagens para a Internet não convém a exibição de imagens de tamanhos de arquivos grandes, pois, as mesmas necessitariam de um tempo maior para o carregamento e exibição em um navegador.

Grande parte das imagens formada por pixels que são apresentadas hoje na Internet são de baixa resolução, para obterem uma visualização mais rápida nos navegadores. Com isso, estas imagens passam a ter uma péssima qualidade de visualização quando a mesma é redimensionada, ou seja, aumentada.

De um outro lado, o SVG apresenta-se em um formato vetorial, guardando da imagem caminhos e formas, ao invés, de *pixels*. O formato vetorial do SVG apresenta-se mais eficiente havendo a necessidade de redimensionamento das imagens, pois, não perde a qualidade da imagem quando esta é redimensionada, além de herda do XML a estrutura de significado e conteúdo da imagem.

Gráficos vetoriais na internet surgiram a partir de dois grupos, o primeiro formado por: Adobe, IBM, Netscape e Sun que submeteram uma proposta a *World Wide web Consortium* (W3C) sobre um documento "XML-based" chamado *Precision Graphics Markup Language* (PGML) em abril de 1998 e o segundo grupo formado por: HP, Macromedia, Microsoft e Visio que também submeteram uma proposta a W3C sobre um "XML-based" chamado *Vector Markup Language* (VML) em maio de 1998. Como resultado destas duas propostas formou-se o grupo SVG, o qual foi publicado em outubro de 1998.

Tendo em vista que grandes empresas se uniram em busca de um formato de imagem diferente para a web, pode-se considerar que há um grande interesse de empresas de tecnologia no formato de gráficos vetoriais, por isso, o SVG já nasce com uma grande expectativa para a construção de gráficos vetoriais, com o principal objetivo

de suprir as necessidades encontradas nos tipo de imagens que se encontram hoje na web, imagens formadas por *pixels*.

É de grande importância lembrar que o formato SVG não pretende substituir as imagens formadas por *pixels* e sim ser uma alternativa para sistemas que utilizem imagens que possam ser transformadas no formato vetorial.

O Padrão SVG tem como principal finalidade descrever gráficos 2D, interativos e animados, em uma página que pode se adaptar ao tamanho e a resolução dos computadores. Estes gráficos no formato SVG podem ser visto tanto em navegadores de computadores pessoais ou na tela de um Palm ou de um telefone celular com acesso a web.

Sendo uma linguagem derivada do XML, o SVG torna-se uma figura gráfica definida e cheia de efeitos, podendo ser adaptada facilmente a um monitor de 18 polegadas de um *desktop* ou a uma minúscula tela de um PDA, permitindo que textos inseridos nas imagens também sejam redimensionados sem nenhum problema para a leitura. Podendo até mesmo estes textos ser detectado por ferramentas de busca ou de tradução. Essas características aumentam a expectativa sobre o formato SVG para cada vez mais ser utilizados em páginas web.

É permitido na linguagem SVG três tipos de objetos gráficos: formas gráficas vetoriais (caminhos que consistem em linhas e curvas), imagens e textos. Estes objetos gráficos podem ainda serem agrupados, transformados e filtrados.

A idéia principal que motivou o SVG foi simples: criar uma solução genérica orientada a documento para gráficos que pudessem ser adaptados para os meios modernos da Internet. A especificação SVG não é somente um formato para gráficos especiais. É uma aplicação que seria projetada por especialistas para adaptar-se aos mais recentes avanços de gráficos 2D.

A seguir é apresentado um exemplo de um código SVG, que desenha um objeto retângulo.

Este retângulo é formado por pontos e não *pixels*, podendo ser redimensionado sem a perda da qualidade, seja em uma tela de um palm ou de um computador *desktop*. A Figura 5.1 ilustra o código de um retângulo.

Figura 5 1 - Utilizando atributo rect.

A primeira linha do código especifica a versão do XML e o caractere encoding do documento.

A segunda, terceira e quarta linhas são opcionais. O elemento *DOCTYPE* refere-se a URL que especifica o local do arquivo que descreve o formato legal do documento.

A quinta linha especifica o início de um documento SVG, com a *tag svg*. A *tag svg* contem dois atributos, *width* que especifica a largura do documento e *height* que especifica a altura. Pode-se observar no código a seguir como se define o valor fixo para um documento.

<svg width=”800” height=”500”>

A sexta linha é de comentários.

A sétima linha contém a *tag g*, que apresenta uma opção chamada *transform*, possibilitando inserir coordenadas no eixo x e y para o objeto, no código apresentado que representa a Figura 5.1, as coordenadas (x, y) possuem o valor 50.

A oitava, nona e décima linha contém a *tag rect*, onde é desenhado o retângulo, os atributos *x* e *y* definem a origem do objeto e as opções *width* e *height* definem a

largura e altura do retângulo respectivamente. Por último a *tag style*, define a cor de preenchimento do objeto.

A décima primeira linha finaliza o elemento g.

A décima segunda linha finaliza o documento *svg*.

5.2- Características gráficas

O SVG pode ser visto como uma solução de desenho, como tal, todas as características primitivas básicas de um vetor gráfico são encontradas no SVG. Estas características incluem objetos gráficos como: linhas, retângulos, polígonos, círculos e elipses.

Além dessas características, o SVG também suporta *paths* (caminhos), incluindo cubos, curvas e arcos elípticos. Cada um destes objetos gráficos tem diferentes características, porém, podem ter a mesma característica de transformação e agrupamento.

No SVG, se pode especificar uma transformação em um objeto com instruções aditivas como: *scale*, *translate* ou *rotate*. Além disso, é possível fazer a formatação das características do objeto através do uso de CSS (*Cascading Style Sheets*)

CSS é um simples mecanismo para a criação de estilos para páginas web como: cores, fontes, espaçamentos, etc.

A seguir são definidos alguns exemplos de códigos utilizando algumas formas básicas do SVG. Todos os exemplos são ilustrados pela Figura 5.2

Utilizando o atributo *polygon*.

```
<svg width="100%" height="100%">
      <g transform="translate(50,50)">
            <polygon
            points="0,0 150,0 150,50 0,50"
            style="fill:red;"/>
</svg>
```

Utilizando o atributo *path*.

```
<svg width="100%" height="100%">
      <g transform="translate(50,50)">
            <path
            d="M0,0 150,0 150,50 0,50"
            style="fill:red;"/>
</svg>
```

Utilizando o atributo *line*.

```
<svg width="100%" height="100%">

      <g transform="translate(50,50)">
            <line x1="0" y1="0" x2="150" y2="0" stroke="red"/>
            <line x1="150" y1="0" x2="150" y2="50" stroke="red"/>
            <line x1="0" y1="50" x2="150" y2="50" stroke="red"/>
            <line x1="0" y1="0" x2="0" y2="50" stroke="red"/>
      </g>

      </svg>
```

Todos os códigos apresentados acima apresentam o mesmo desenho quando exibidos no navegador, um retângulo como o da Figura 5.2, apesar de utilizarem atributos diferentes (*polygon*, *path* e *line*).

Figura 5 2 - Utilizando o atributo polygon, path e line.

O SVG é organizado em três estruturas, através de herança, grupo e estrutura, que oferecem flexibilidade para a transformação e a modelagem dos objetos. Por exemplo, pode-se dimensionar um grupo de três objetos gráficos diferentes com uma única instrução de transformação. Do mesmo modo, se podem preencher todos os três objetos com uma única cor, usando uma única instrução de modelagem no nível do grupo. É possível também dar a cada objeto diferentes identificações XML, logo se pode referenciá-los facilmente em outras partes do documento.

O SVG também oferece capacidades gráficas avançadas como suporte para gradientes, transparência e filtros. É possíveis o preenchimento de objetos com uma cor

ou com gradientes linear e radial, além da possibilidade de definir filtros para cada um dos objetos.

Além das opções do formato de vetor gráfico, o SVG também suporta imagens *rasters* e textos. Os formatos JPEG e PNG são suportados e podem ser incluídos em um documento como arquivos internos ou externos.

Uma outra característica do SVG trata dos dispositivos portáveis chamada SVG Mobile, tornando-se possível o envio de mensagens animadas, coloridas e com som.

Algumas outras características de SVG são descritas abaixo:

- Código compacto e portável.
- Imagens complexas podem ser criadas usando escalonamento e rotações
- Fácil geração para a Internet.
- Podem ser geradas por Java Servlets e JSP.
- Alta resolução e alta performance quando redimensionada.
- Não há perda de resolução quando as imagens são redimensionadas.

Em SVG é possível também a transformação de arquivos textos em documentos SVG por vários de caminhos, como: XSLT, linguagens de Script (Perl, Python, Ruby) e script Unix.

5.3 - Características dinâmicas

Embora o SVG já ofereça avançadas capacidades gráficas 2D, este vai ainda mais longe que os gráficos estáticos oferecendo capacidades dinâmicas extensivas. O dinamismo do SVG é representado por animações e *scriptings*.

5.3.1 - Animação

A animação é quem faz o maior uso de vetores gráficos hoje na Internet, por isso o SVG oferece também ferramentas para esta tendência. A animação em SVG pode-se dar por diferentes métodos como: pelo elemento *animation*, pelo SVG DOM e pelo *Synchronized Multimedia Integration Language* (SMIL).

A animação em SVG foi desenvolvida em colaboração do W3C com o SMIL.

Na sua segunda versão, o SMIL introduziu um módulo de animação para ser reutilizado e adaptado em qualquer outra linguagem XML que precisaria deste tipo de funcionalidade. SVG aproveitou-se disso e incluiu todas as funcionalidades definidas pela

Abaixo segue um exemplo de código para animação de um retângulo em SVG.

```
<svg width="100%" height="100%">
<g transform="translate(50,50)">
  <rect x="0" y="0"
        width="100" height="50"
        fill="red">
        <animateTransform attributeName="transform"
                          attributeType="XML"
                  begin="0s" dur="4s"
                          fill="freeze"
                          from="0" to="400"/>
  </rect>
</g>
</svg>
```

Será considerada a explicação do código acima a partir do atributo *animateTransform*, onde o atributo *atrributeName* é o que recebe um nome para a animação, o *attributeType*, pode ser XML ou CSS, o elemento *begin* determina o tempo para dar inicio a animação, o elemento *dur* define o tempo de duração da animação, a atributo *fill="freeze"* determina que o objeto fique desenhado no lugar de parada, o atributo *from* define o ponto de partida do objeto e o atributo *to* define o ponto de parada do objeto.

A Animação em SVG fornece o mesmo tipo de capacidade de animação que o Flash tem usado. Entretanto, a animação em SVG é mais flexível e oferece controle sobre a vida da animação.

5.3.2 - Scripting

Gráficos SVG podem refletir sobre qualquer tipo de interação através de *scripting*. Comumente, aplicações SVG oferecem um interpretador ECMAScript que deixa acessar, ler e escrever o seu documento.

Por SVG ser baseado em XML, todo o conteúdo de um documento SVG poderia tecnicamente ser "scriptado" com operações de strings. Entretanto, isto seria um método um pouco cansativo de ser usado. Para evitar este método cansativo, o SVG conta com uma outra tecnologia XML, o DOM. DOM é uma API designada para acessar e manipular o conteúdo de um XML parsed (XML analisado). Além de ser um modelo orientado a objeto e que diminui a tarefa de analise continua.

- DOM

Primeiramente, o DOM forma a base para a manipulação de documentos. Esta API nos permite a navegação através da árvore do SVG, acesso a elementos particulares e troca de valor de um atributo.

É importante ressaltar que o objetivo principal do DOM não é simplesmente permite a interação cliente-servidor com o documento. É um documento completo de criação de solução. Desenvolvedores conseguem usar a mesma API no lado do servidor para criar documentos SVG. Sendo capaz de ser feito no servidor, significa também que pode ser utilizar o conteúdo SVG no lado do cliente, desta forma oferecendo para um experiente usuário e reduzindo o trafego na rede na aplicação cliente-servidor.

É importante saber que se pode usar DOM para estender um conjunto de características SVG. DOM faz isto fácil criando componentes reutilizados, para um

novo objeto gráfico de nível mais alto para distribuir com clientes ou com os desenvolvedores.

5.4 - Serviços em um ambiente gráfico distribuído usando SVG

Este tópico descreve o ambiente construído para este trabalho que tem como principal finalidade à geração de objetos e gráficos no formato SVG.

Para a geração dos objetos e gráficos no formato SVG o ambiente desenvolvido propõe a implementação de um sistema a partir da estrutura Cliente-Servidor e tem a sua construção baseada no modelo MVC em conjunto com a tecnologia para a web semântica, o formato gráfico SVG.

O serviço construído dá suporte tanto para um computador pessoal, como para dispositivos sem fio como: PDAs e telefones celulares, já que, as tecnologias utilizadas oferecem suporte para isso.

A utilização do modelo MVC como base para o sistema desenvolvido se deve pela maneira como este organiza o desenvolvimento e a construção de sistemas para a web, separando a implementação do sistema de sua apresentação, além de utilizar a linguagem Java como base dos componentes que formam este modelo.

O modelo MVC como já descrito neste trabalho é formado por Servlets, Bean e JSP. Os Servlets são importantes para tratar à lógica do negocio, neste trabalho tem algumas funções como tratar requisições e direcioná-las corretamente, além de chamar a construção de beans e também a conexão com o banco de dados. Os Beans são utilizados para a criação, remoção e atualização de um objeto, além de permitirem e caracterizarem o sistema na forma de poder ser acessado por vários usuários ao mesmo tempo. E as páginas JSP são utilizadas principalmente para a visualização do sistema em conjunto com o formato SVG, além da possibilidade de acesso aos beans para a recuperação dos mesmos.

A linguagem de programação Java possibilita que o modelo baseado se beneficie de suas diversas vantagens de linguagem orientada a objetos sobre outras linguagens de programação não orientadas a objeto.

O formato SVG se tornou fundamental na implementação em conjunto com este modelo, pois, possibilita a construção de imagens vetoriais, ou seja, não distorcem a medida em que se redimensiona a imagem, além de permitir um carregamento mais rápido do que imagens hoje apresentadas na Internet, que são as imagens formadas por *pixels*.

Um outro item importante a se ressaltar da importância do formato SVG em relação a imagens *rasters* se deve ao significado semântico deste formato, possibilitando que este tipo de imagem não tenha apenas características de formatação como: cor e tamanho, mas também um conteúdo com significado. Este conteúdo com significado possibilita que este tipo de imagem possa facilmente ser encontrado por mecanismos de buscas da Internet, além de permite que seja reconhecido por qualquer sistema ou base dados que utilize a linguagem XML.

O ambiente desenvolvido funciona para a web e qualquer usuário final pode ter acesso desde que esteja conectado a Internet.

Todos os objetos criados são armazenados no banco de dados do sistema. Neste sistema é utilizado o Postgres. A escolha do Postgres em relação a outros bancos se deu devido alguns fatores como: atende perfeitamente aos recursos necessários ao sistema, fácil integração com a linguagem Java, grande material sobre o banco disponível na Internet e em livros, facilidade na instalação e no aprendizado.

Não houve comparação na utilização do sistema com outro banco de dados, pois, não é este o foco principal do trabalho.

O servidor utilizado para hospedar o sistema escolhido é o jakarta tomcat 5.5.9. Houve também o estudo do servidor Jboss, porém, o tomcat mostrou-se suficiente e supriu todas as necessidades requisitadas para a construção do sistema.

A geração de gráficos por este software permite que o usuário final construa interfaces para páginas web, desenhos e artes, assim como gráficos 2D, utilizando-se e se beneficiando da característica do formato SVG, como: vetorização e semântica, além de um carregamento mais rápido nas páginas a Internet, conforme comprovado no tópico 5.5 deste capítulo.

A utilização das tecnologias citadas para a construção deste sistema possibilita a independência do sistema em relação à plataforma, podendo o serviço ser utilizado por um sistema Windows, Unix ou Macintosh.

CAPÍTULO 6 – SISTEMA DESENVOLVIDO BASEADO NO MODELO MVC

6.1 - Funcionamento do sistema de login e cadastramento de usuários

Tudo que é gerado ou criado pelo usuário pode ser salvo pelo próprio usuário no computador, além da possibilidade de envio dos objetos, artes e gráficos gerados para uma área reservada a cada usuário do sistema.

Esta área reservada ao usuário é criada no momento em que o usuário se cadastra no ambiente. Para o cadastro são necessários os preenchimentos de alguns campos obrigatórios como: nome, sobrenome, email, usuário e senha.

O sistema não permite o cadastro do campo usuário repetido, ou seja, caso se tente cadastrar um usuário já cadastrado, o sistema envia o usuário a uma página erro com a mensagem de usuário já cadastrado e uma opção para um novo cadastramento.

Após o usuário ter realizado o seu cadastro e o mesmo ser cadastrado com sucesso, este é enviado para a página de login do sistema. Nesta página o usuário deve preencher os campos login e senha conforme inserido no cadastro, caso tenha sucesso no login, o usuário fica habilitado a utilizar o sistema, caso haja erro de usuário ou senha, o usuário então é enviado para uma página de erro, e tem opções como: tentar novamente o login ou fazer o cadastro para um novo usuário.

A seguir pode ser visto na Figura 6.1 o diagrama que ilustra o funcionamento do sistema de login e cadastro para o ambiente desenvolvido. É importante ressaltar que já para este sistema de login e cadastro, a criação da área para cada usuário e login é utilizado o modelo MVC para o desenvolvimento.

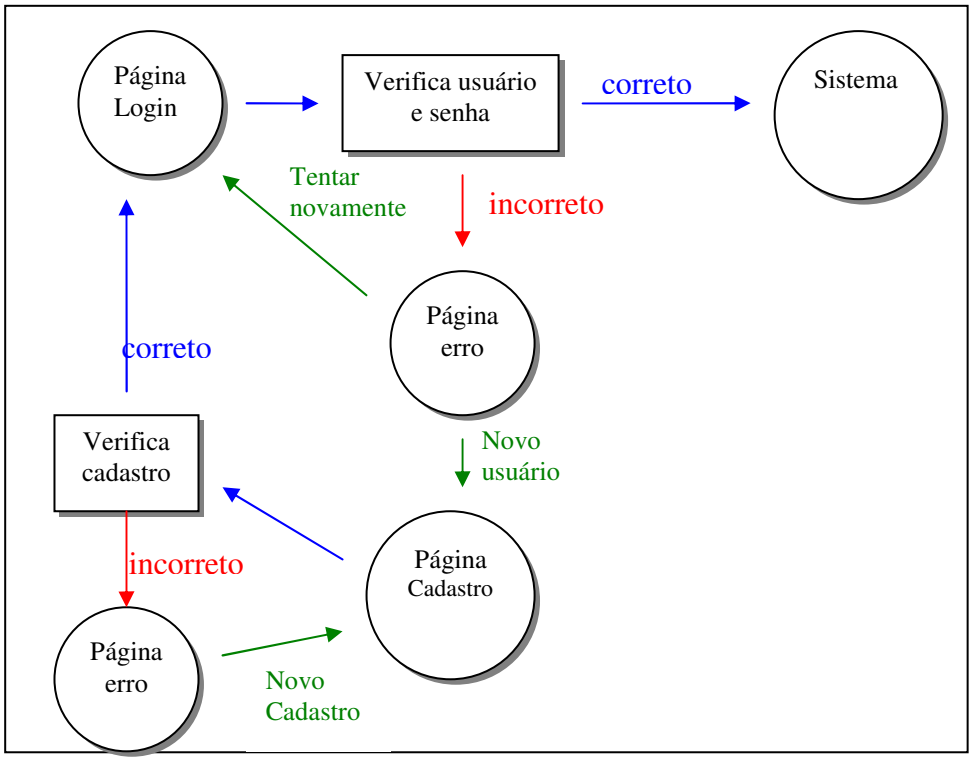

Figura 6 1 - Diagrama do Sistema de Cadastro e Login.

Para completar a total descrição do sistema de login a Figura 6.2 ilustra a conexão do sistema com o banco de dados. A função do banco de dados é reconhecer quando um usuário já está cadastrado e quando o usuário ou senha do usuário não estão de acordo com os dados preenchidos no cadastro.

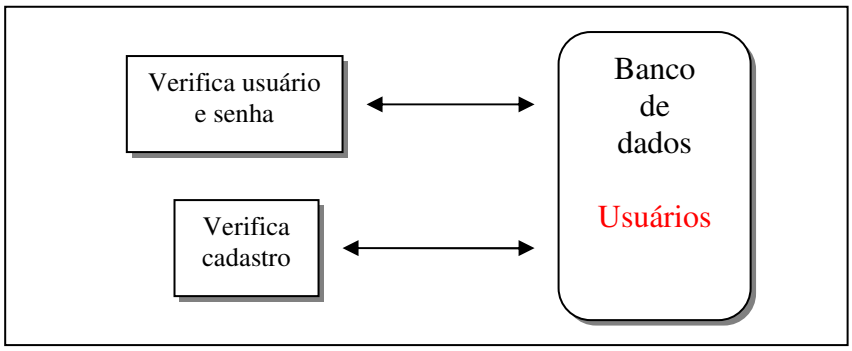

Figura 6 2 - Conexão do Sistema de Login com o banco de dados.

A conexão do sistema com o banco de dados é realizada a partir do envio dos dados preenchidos na página de cadastro. Estes dados são enviados ao Servlets que tem a função de chamar o bean para a criação do usuário. Após a criação do usuário, o bean é chamado pelo Servlets que tem a função de cadastrar o usuário no banco de dados.

Realizado o cadastramento o Servlets então chama a página JSP que exibe a mensagem: “cadastramento realizado com sucesso”.

A Figura 6.3 baseada no modelo MVC ilustra os passos a seguir para o cadastramento de um usuário.

1. Preenchimento do formulário pelo usuário
2. Os dados preenchidos são enviados para o Servlets controlador.
3. O Servlets controlador chama o Bean Usuário para a criação do usuário.
4. O Bean criado é retornado ao Servlets.
5. O Bean criado é inserido no banco de dados, caso o código de acesso ao sistema não seja repetido.
6. O Bean é retornado ao Servlets.
7. O Servlets chama a página JSP de confirmação do cadastro ou de usuário já cadastrado.
8. A Página JSP é exibida ao usuário.

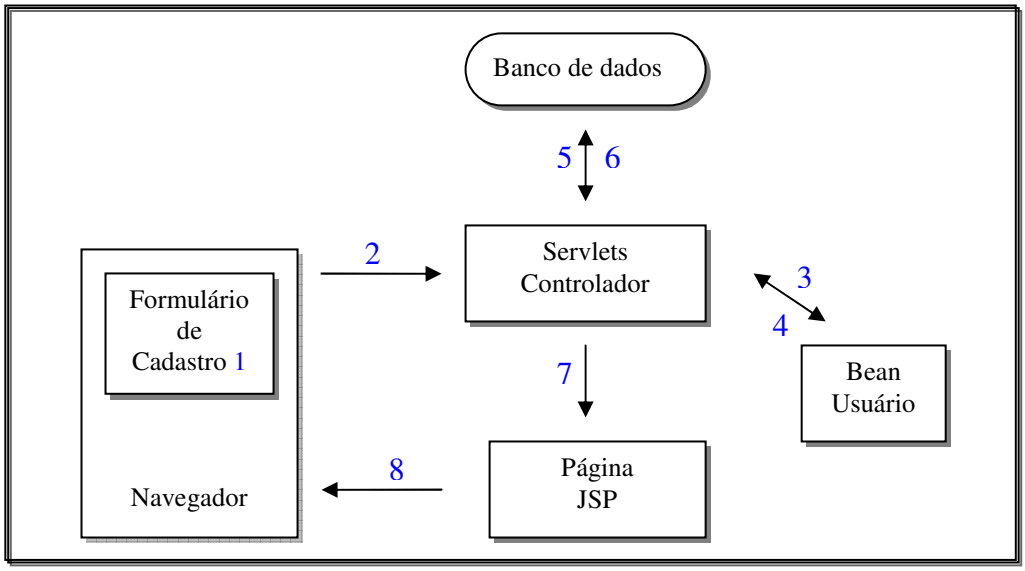

Figura 6 3 - Modelo para Cadastro de usuários

Após a entrada no sistema o usuário pode utilizar todas as funções deste ambiente, como: criação de círculos, retângulos, elipses, linhas e gráficos no formato 2D.

6.2 - Funcionamento do sistema

A arquitetura proposta para o sistema é a de um servidor para múltiplos clientes. Os clientes são responsáveis por enviar as informações ao servidor, que tem a responsabilidade de oferecer os serviços e atender a todos os clientes simultaneamente.

Além disso, no servidor devem ficar cópias atualizadas de cada documento. Quando um cliente atualiza um documento o mesmo é passado para o servidor para atualização. O servidor tem o dever de atualizar o documento seja o mesmo da área restrita ou o documento da área compartilhada.

Na Figura 6.4 pode-se visualizar o funcionamento de uma atualização do sistema, por um Cliente A.

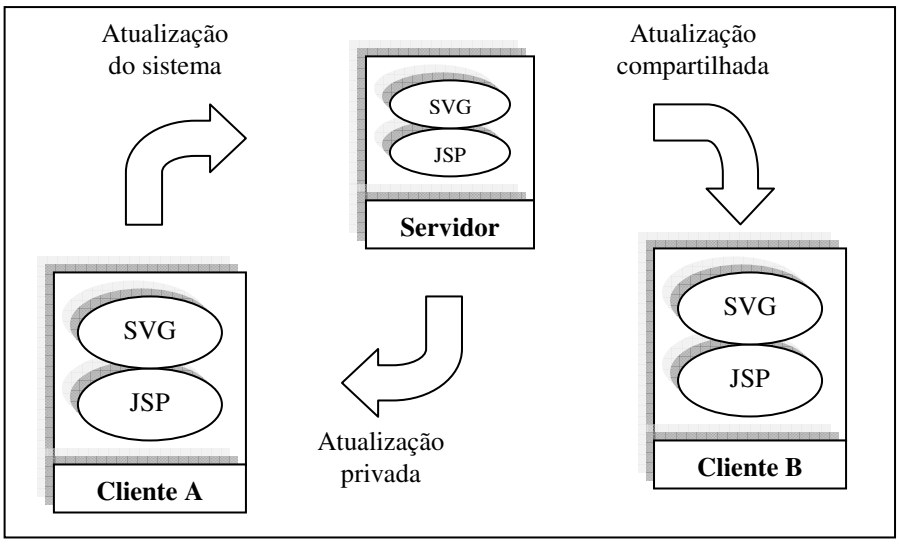

Figura 6 4 - Atualização do Sistema por um Cliente A.

Sempre que o Cliente A atualiza seus dados no servidor, o servidor tem o dever de atualizar os mesmos dados nos outros clientes caso a atualização seja a área

compartilhada, que é a área de acesso a todos os clientes, caso contrario, apenas a área do cliente A será atualizada.

Para ter acesso ao servidor e poder manipular os dados armazenados no mesmo, basta que o cliente tenha instalado na sua máquina um plug-in SVG, como o Adobe SVGView.

Funções de cada cliente são destacadas a seguir.

- Cada cliente deve estabelecer conexão com o servidor;
- Armazenar copias locais de documentos;
- Enviar copias atualizadas para o servidor;

Do lado servidor, encontram-se documentos personalizados de cada usuário, disponíveis para que o cliente possa consultá-los, modificá-los e trocarem informações.

Utilizando o servidor Jakarta-Tomcat para tratamento de arquivos específicos JSP, os serviços disponíveis tornam-se completamente dinâmicos, permitindo que os usuários possam personalizar seus documentos da maneira mais apropriada.

Funções do servidor são destacadas a seguir.

- O servidor deve esperar pela conexão de um cliente e lhe oferecer um identificador único;
- Manter as versões de cada documento atualizadas;
- Aplicar atualizações nos cliente para os documentos compartilhados;
- Enviar todas as atualizações para cada cliente conectado;
- Enviar documentos atualizados quando os mesmos foram requisitados pelos clientes.
- Responsável por toda a gerência do sistema

O usuário ainda tem opções como cadastrar e remover um objeto da página atual e ainda a opção de atualização da posição, cor, borda, largura, altura, tamanho e raio de acordo com as características de cada objeto.

A Figura 6.5 mostra o diagrama de caso de uso para o sistema desenvolvido da perspectiva do usuário.

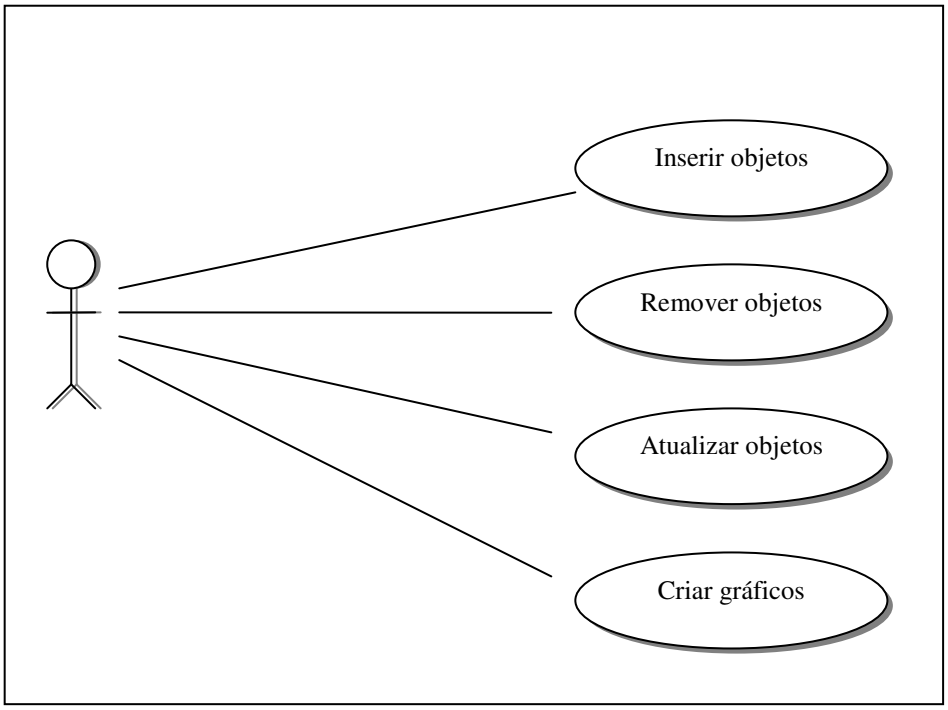

Figura 6 5 - Diagrama de caso de uso para o Sistema da perspectiva do usuário

O modelo de funcionamento para o sistema desenvolvido pode ser visto da Figura 6.6, a relação entre os componentes do modelo MVC em conjunto com o formato SVG e também a relação do sistema com o banco de dados.

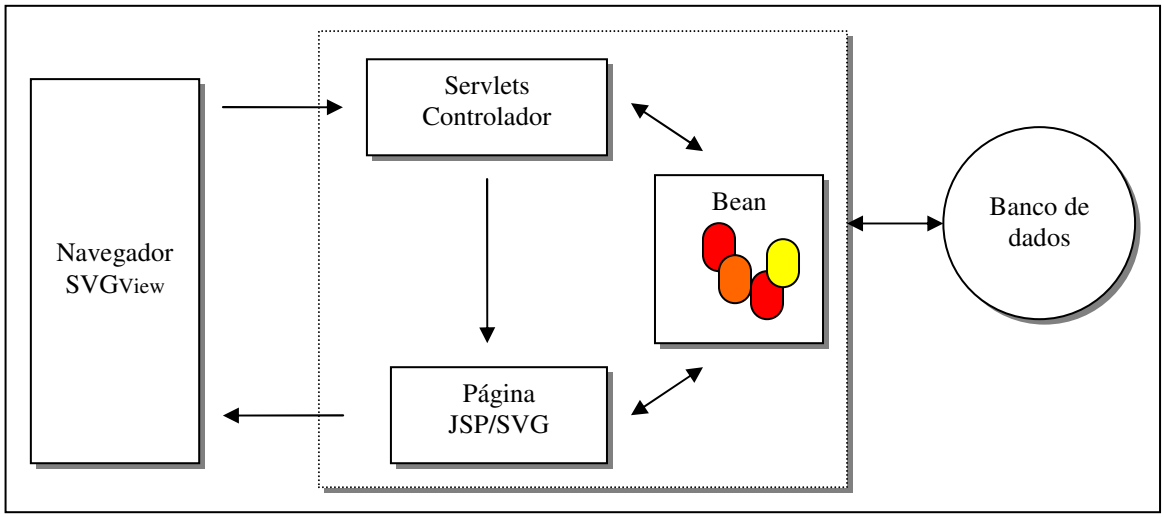

Figura 6 6 - Modelo de funcionamento do sistema desenvolvido

O Controlador é um Servlets que tem a responsabilidade de identificar que tipo de ação foi requisitada pelo usuário para poder então tomar a sua decisão de criação da tarefa a ser realizada, que pode ser: dinâmica ou estática. Caso seja uma tarefa estática, a tarefa é realizada e passada diretamente para a Visualização que repassa ao cliente o objeto SVG. Caso a tarefa seja dinâmica, então será criada uma instancia de um modelo do objeto e da ação sobre o objeto que foi requisitado, logo, o mesmo será passado também a Visualização para a criação do objeto SVG.

O Modelo é representado por Beans para a construção dos objetos requisitados pelos clientes.

A Visualização é a parte de visualização de objetos gerados pelo Modelo que são visualizados pelo cliente em SVG.

Um exemplo para a utilização do sistema é descrito a seguir.

Quando um usuário solicita a criação de um retângulo o mesmo é enviado de uma página principal do usuário para o Servlets que analisa a solicitação realizada. Após a analise feita pelo Servlets, que define qual objeto deve ser criado ou o tipo de função a ser realizada, este então chama o bean para a criação do objeto solicitado. Após a criação do objeto, este é chamado pelo Servlets que o redireciona ao usuário que

o requisitou. Este redirecionamento é feito à página principal do usuário que se encarrega de visualizar o objeto requisitado.

São apresentados a seguir os aspectos principais do código desenvolvido, mas as partes principais dos Servlets, beans e JSP/SVG.

- Código para a criação do retângulo pelo usuário para depois enviá-lo ao Servlets.
- Código do Servlets que recebe os dados enviados pelo usuário.
- Código do Servlets que chama o Bean.
- Código para a criação do bean.
- Código do Servlets que insere o bean no banco de dados.
- Código que direciona a requisição a página JSP/SVG.
- Código SVG exibido para o usuário.

6.3 - Relação do sistema com o banco de dados

Todos os objetos de cada usuários são armazenados no banco de dados.

No banco de dados existem tabelas para o armazenamento, remoção e alteração dos dados armazenados por cada cliente. Pode-se nomear algumas tabelas como: tabela usuário, responsável pelos dados cadastrados com informações do usuário. Tabela retângulo, responsável por armazenar todas as características enviadas pelo usuário para a criação de um objeto retângulo. Estas características podem ser: largura, altura, cor, posição na tela e outras, além do usuário responsável pela criação do retângulo.

A tabela usuário tem os seguintes campos: `codigou`, senha, nome, sobrenome e email. O campo `codigou` é reservado para armazenar um identificador único para cada usuário, além de ser o responsável por evitar que um `codigou` já existente na

tabela usuário seja cadastrado. Todos os outros campos podem ser cadastrados sem restrição alguma.

A tabela retângulo relaciona-se com a tabela usuário pelo campo `codigou`. A cada criação de um objeto por um usuário qualquer, o objeto é armazenado no banco de dados com a informação do usuário que o criou, permitindo assim, a identificação de todos os objetos retângulos criados e quais usuários os criaram.

Para uma consulta no banco de dados para saber quais objetos retângulos pertencem ao usuário 15, basta comparar o `codigou` da tabela usuário que me indica o usuário a ser consultado e o `codigou` da tabela retângulo que me indica que usuário criou o retângulo.

A seguir pode ser visto o código resumido que retorna a consulta de retângulos para um usuário com código de acesso igual a 15.

```
int codigo = 15;
Usuario u = (Usuario) guestListu.get(l);
Rect r = (Rect) guestList.get(i);
    if( (u.getCodigou()==codigo) && (r.getCodigou()==codigo))
    {
          <g>
                      <rect x="<%=r.getX()%>"
                      y="<%=r.getY()%>"
                      width="<%=r.getLargura()%>"
                      height="<%=r.getAltura()%>"
                      style="fill:<%=r.getCor()%>"/>
          </g>
    }
```

A seguir é visto na Figura 6.7 o diagrama resumido e a relação das tabelas do banco de dados.

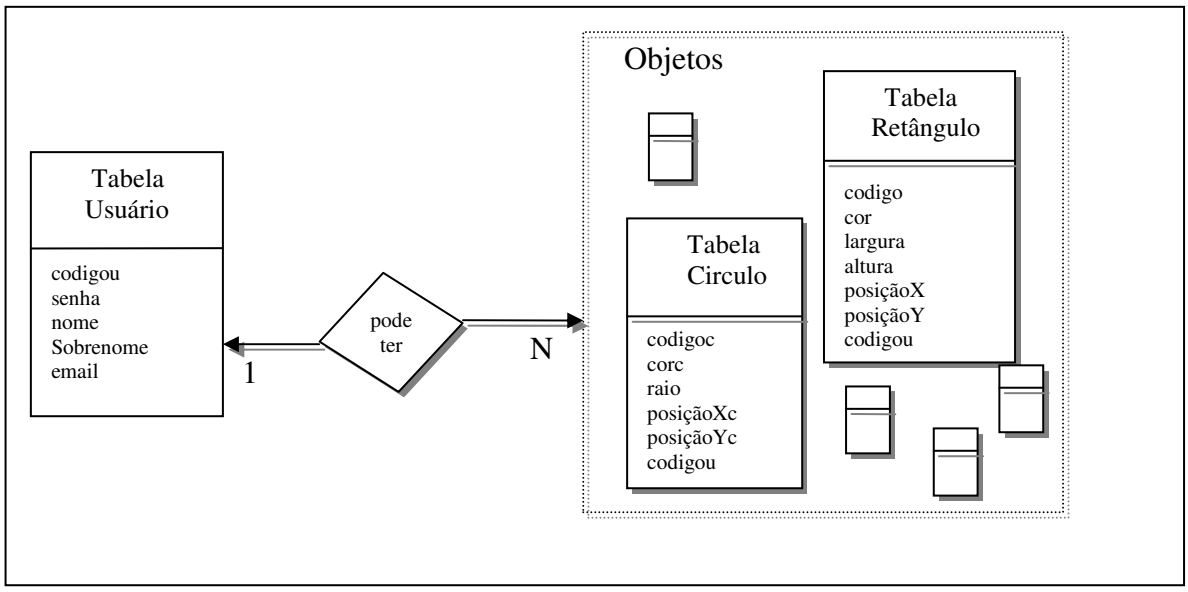

Figura 6 7 - Diagrama da relação entre as tabelas do banco de dados.

6.4 – Criação de interfaces e gráficos com o sistema

O modelo SVG é utilizado na geração de interfaces e gráficos pelo sistema por ser uma linguagem derivada do XML, possuindo assim, a grande vantagem de ser uma linguagem de significado, ou seja, que dar significado aos seus elementos.

A interface SVG construída disponibiliza gráficos e objetos no formato SVG que podem ser dinamicamente alterados a cada acesso de um cliente.

A interface SVG a ser construída oferece aos usuários a possibilidade de interagir com ferramentas de desenho; atualizar o modelo com estes elementos; interpretação de comandos do usuário para geração de elementos SVG; mostrar e atualizar documentos do modelo SVG.

Para a criação de gráficos está disponível a opção de gráficos em colunas, que pode se expandir ainda para varias outras opções como: barras, linhas, pizza, etc.

Para a criação dos gráficos o usuário tem a necessidade de saber quantas linhas e colunas serão preenchidas. Após a escolha do número de linhas e colunas o usuário deve preenchê-las, sabendo-se que: a primeira linha deve ser preenchida pelos títulos

que irão gerar a legenda e a primeira coluna pelos títulos das colunas geradas pelo gráfico.

A Figura 6.8 ilustra uma tabela com os dados para a construção do gráfico da Figura 6.9 gerado pelo sistema.

Alunos	Bim1	Bim2
João	6	9
Maria	10	8
Pedro	8	7

Gerar Gráfico

Figura 6 8 - Tabela para a geração de um gráfico.

A Seguir segue o gráfico da Figura 5.11.

Figura 6 9 - Gráfico gerado pelo sistema

O objeto gráfico apresentado na Figura 6.9 pode ser perfeitamente visualizado em telas de dispositivos moveis, assim como, em uma tela de um monitor de um computador pessoal. Detalhes do gráfico são vistos em perfeita visualização sem a perda da qualidade do mesmo.

6.5 - Avaliação primária de desempenho (HTML raster x SVG)

O Sistema desenvolvido detalha os tempos obtidos quando são requisitados objetos em SVG e os compara com tempos obtidos quando solicitadas imagens no formato rasters JPG.

Para imagens de tamanho pequeno, a comparação entre os formato rasters e o formato SVG tem como tempo de resposta para abrir uma imagem em um navegador é quase insignificante, porém, é preciso entender que imagens com o mesmo tamanho e tempo de respostas iguais podem ser tornar diferentes quando é observado o formato da imagem.

O formato SVG tem a vantagem de aumentar a visualização da imagem e não distorcê-la, ao contrario do formato rasters.

No exemplo analisado pode-se observar a mesma imagem em formatos JPG e o formato vetorial SVG, em um primeiro momento a definição parece não fazer diferença, pois, como pode se comprovar nas Figuras 6.12 e 6.13, os formatos se equivalem no que se diz respeito à visualização da imagem.

É importante analisar também os tempos de respostas obtidos para estas imagens. O formato SVG mostrou-se mais rápido que o formato JPG analisado conforme o gráfico apresentado a seguir.

O Gráfico foi construído com base na medida de tempo de um arquivo tanto no formato JPG como no formato SVG de tamanho aproximado de 250 kb. Todos os outros tempos são estimativas que se baseiam no modelo analisado.

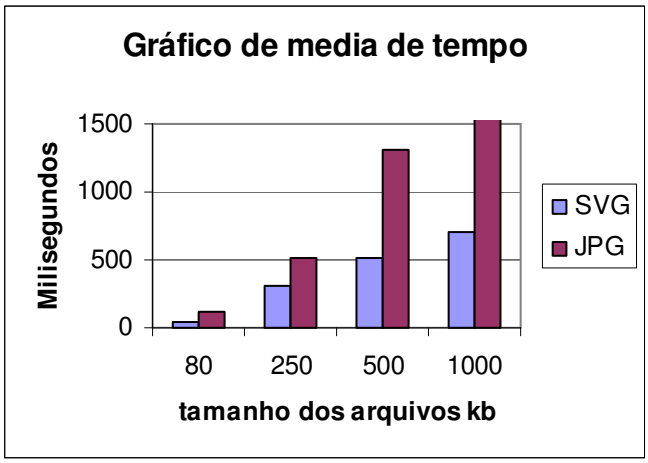

Figura 6 10 - Gráfico de comparação entre formatos

Com base no gráfico da Figura 6.10 ilustra as vantagens em tempo de carregamento de um imagem no fomato SVG em relação ao formato rasters JSP.

Uma outra vantagem de se trabalhar com imagens no formato SVG pode ser analisada a partir das Figuras 6.12 e 6.13.

São Figuras que possuem o mesmo poder de visualização quando estão em seu estado original, não se notando diferença alguma.

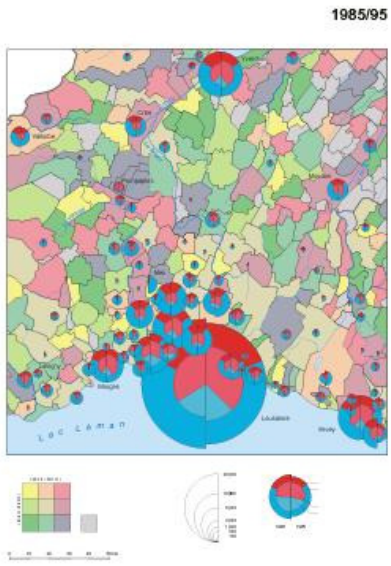

Figura 6 11 - Imagem JPG

Porém quando se redimensiona a imagem para um maior detalhamento da mesma é possível notar a grande diferença entre os dois formatos analisados. A Figura

6.12 apresenta o formato rasters JPG, enquanto que, a Figura 6.13 apresenta o formato SVG com a mesma parte aumentada para analise.

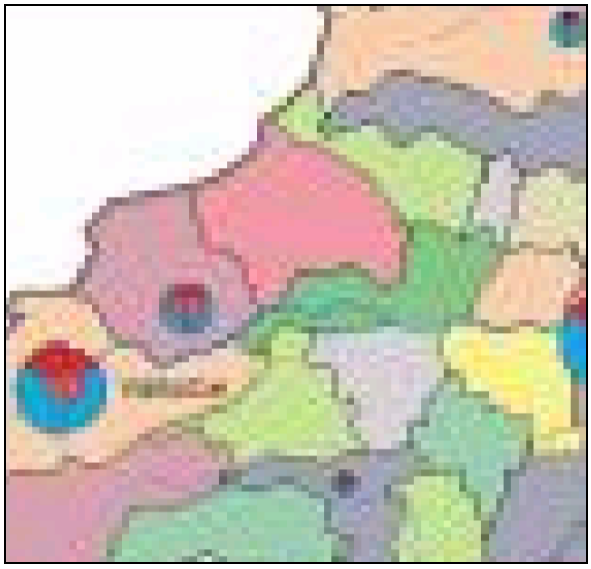

Figura 6 12 - Imagem JPG aumentada.

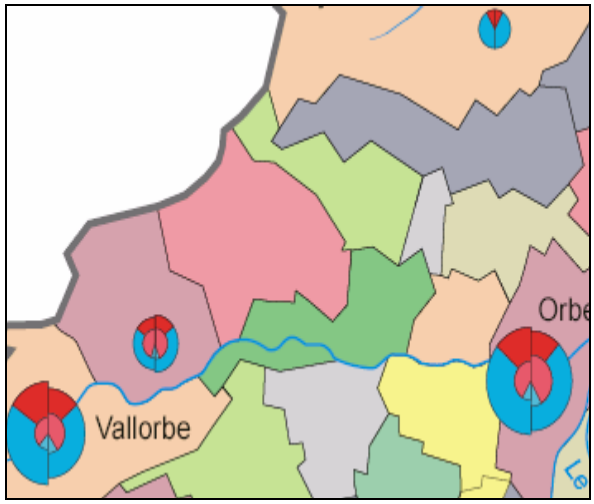

Figura 6 13 - Imagem SVG aumentada.

Algumas vantagens de visualização são possíveis de serem notadas como:

- Melhor visualização da imagem.
- Não há perda de qualidade.
- Pode-se observar melhor o texto definido na imagem.
- A imagem tem significado semântico.

Com certeza após os estudos realizados e resultados obtidos é possível observar porque grandes empresas voltadas para o desenvolvimento de software para a web e serviços oferecidos para clientes estão cada vez mais adotando o formato SVG para o desenvolvimento de interfaces, gráficos e animações para a web.

No próximo capítulo são apresentadas as conclusões finais para este trabalho juntamente com a expectativa para este ambiente desenvolvido.

CONCLUSÃO

O estudo realizado bem como a proposta de criação de um sistema baseado em uma tecnologia da web semântica permite que o ambiente apresentado tenha vantagens sobre outros sistemas existentes, que tem como objetivo a geração de interfaces para a web. Estas vantagens podem ser consideradas pelas tecnologias utilizadas juntamente com o modelo de construção do sistema, além da integração no modelo de uma linguagem semântica, permitindo assim o reconhecimento de cada objeto criado, pois, são definidos semanticamente.

Uma outra vantagem a ser apresentada é a realização da construção das interfaces pelo sistema diretamente na Internet, sem a necessidade de ser instalar o programa proposto por este trabalho no computador do cliente.

O sistema ainda se beneficia por ser baseado no modelo MVC, separando o desenvolvimento em partes, apresentação e implementação. Além de permitir que o cliente possa interagir de forma dinâmica com suas interfaces.

O SVG integrado ao modelo de desenvolvimento permite que os objetos e gráficos criados sejam oferecidos sem a preocupação por parte de quem esta desenvolvendo da perda de qualidade de definição dos mesmos quando redimensionados, pois, o formato SVG é vetorial.

A potencialização da web semântica com os servlets e beans que disponibilizam nos clientes os dados e facilitam a interação e também diminuem o fluxo com o servidor, pois, os beans que são dados persistentes que facilitam isto. Além da utilização de uma linguagem derivada do XML, o SVG, que possibilita que futuramente o sistema possa interoperar com outros sistemas.

Pode-se considerar também o tempo de resposta para os objetos criados pelo sistema em relação a imagens rasters. Para imagens pequenas é sensível o ganho em

relação à resolução, porém, na medida que o tamanho e a complexidade da imagem crescem a mesma passa a ter vantagens nos dois sentidos, em resolução e tempo, uma vez que os gráficos vetoriais.

Dificuldades encontradas para o desenvolvimento deste projeto podem se resumir nos poucos trabalhos encontrados na área para uma pesquisa mais detalhada.

Espera-se que este ambiente futuramente possa ser complementado e aprofundado, a partir de seus estudos e testes para que possa ser uma referência no desenvolvimento de interface e gráficos para a web.

REFERÊNCIAS

Adobe's SVG plug-in **.** Disponível em: **<http://www.adobe.com/svg>** . Acesso em 12 de Março de 1995.

ALONSO, Ronaldo. **Sistemas Distribuídos**. Disponível em:<http://www.ronaldoalonso.hpg.ig.com.br/research/sist_distr.htm>. Acesso em 16 de Março de 2005.

Apache Jakarta Projet, Disponível em:<http://jakarta.apache.org/tomcat/>. Acesso em 10 de março de 2005.

Applications Programming in Smalltalk-80(TM):
How to use Model-View-Controller (MVC), Disponivel em: < http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>, Acessado 26 de Janeiro de 2006.

BERNERS-LEE, TIM, HENDLER, James e LASSILA, Ora. **The Semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities**. Disponível em: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>. Acesso em 15 fevereiro 2005.

BURBECK, Steve. **Applications Programming in Smalltalk-80(TM):**
How to use Model-View-Controller (MVC). Disponivel em: **<** http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html >. Acesso em: 10 de fevereiro de 2006.

BUSCHMANN, Frank, MEUNIER, Regine, ROHNERT, Hans, SOMMERLAD Peter, Machael Stal. Pattern-Oriented Software Architecture: A System of Partterns.

CAI, Jason. KAPILA, Ranjit. PAL, Gaurav. - HMVC: The layered pattern for developing strong client tiers. Disponivel em: <http://www.javaworld.com/javaworld/jw-07-2000/jw-0721-hmvc.html>. Acesso em 12 de Maio de 2006.

CAMBIUCCI, W. e RUGGIERO, W. V. **Monitores transacionais** – Construção de um servidor web com enfoque transacional.

CHAMPION, FERRIS, LONA & ORCHARD, 2002 - **web Services Architecture**. Disponível em: < http://www.w3.org/TR/2002/WD-ws-arch-20021114/#whatisws >. Acesso em 15 Outubro de 2005.

CORREIA, Vasco M, Serialização de dados de imagens médicas usando troca de mensagens, Trabalho de iniciação Científica, UNIVEM: Marilia, 2005.

DO-HYUN, KIM e MIN-SOO, KIM. XML and Inter-Operability in DIstributed GIS. Republic of Korea.

ECKHOUSE JR., R.H et alii **– Issues in Distribuited Processing** – Na Overview of Two Workshops. Computer, 11(1):22-26, jan. 1978.

FIELDS, Duane K. e KOLB, Mark, A. - **Desenvolvendo na web com Java Server Pages**, editora Ciência Moderna.

FLYNN, M. J. **Some Computer Organizations and their Effectiveness**. IEEE Transactions on Computers, vol C-21, pp. 948-960, Setembro, 1972.

FOX, Geoffrey, BULUT, Hasan, KIM, Kangseok, KO, Sung-Hoon, LEE, Sangmi, OH, Sangyoon, PALLICKARA, Shrideep, QIU, Xiaohong, UYAR, Ahmet, WANG, Minjun, WU Wenjun. **Keynote Speech:** Collaborative web Services and Peer-toPeer Grids. Indiana University.

HALL, Marty, BROWN Larry. **Core Servlets and JavaServer Pages** – Core Technologies.

IBM SVG viewer. Disponível em: <http://www.alphaworks.ibm.com/tech/svgview>. Acesso em 19 fevereiro de 2005.

JAMES, Clark XP XML Parser. Disponível em: <http://www.jclark.com/xml/xp/>. Acesso em 19 fevereiro 2005.

JAVA TUTORIAL - The Java Tutorial: Thread synchronization, Disponivel em: <http://java.sun.com/docs/books/tutorial/essential/threads/synchronization.html>, Acessado em 25 de Abril de 2005.

JavaWorld: MVC meets Swing. Disponível em: <http://www.javaworld.com/>. Acesso em: 20 fevereiro de 2005.

JavaWorld: Observer and Observable and the MVC Design Pattern. Disponível em: < http://www.javaworld.com/javaworld/javatips/jw-javatip29.html>. Acessado em: 30 de Janeiro de 2006>

KIRNER, Cláudio, Mendes, Sueli B. T. – **Sistemas Operacionais Distribuídos** – Aspectos Gerais e Análise de sua Estrutura, editora Campus Ltda.

LEE, Sangmi, Ko Sunghoon, FOX, Geoffrey, KIM, Kangseok, OH Sangyoon. A web Service Approach to Universal Accessibility. Indiana University.

MENÉNDEZ, Andrés. Uma ferramenta de apoio ao desenvolvimento de web Services.

Microsoft .NET. Disponível em <http://www.microsoft.com/brasil/dotnet/introducao/webservices.asp>. Acessado em 5 março 2005.

MOEN, Sven. - **Drawing Dynamic Trees** , IEEE Software, July 1990, pp. 21-28

MOURA, Daniel, SILVA, Lena, NERY, Marcelo, RODRIGUES, Paulo. Recuperação de Imagens Baseado em Conteúdo. Universidade Federal de Minas Gerais.

NETWORKS, Blaz. Formato SVG promete concorrência ao Flash. Disponível em: < http://www.blazhost.com.br/~forumbl/showthread.php?p=27940 >. Acesso em 18 de Novembro de 2005.

PALLICKARA, Shrideep e FOX, Geoffrey. NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Indiana University.

PEEBLES, R & MANNING, E. – **System Architecture for Distributed Data Management.** Computer, 11(1):40-47, jan. 1978.

PERPÉRTUO, Enio e JAGIELLO, Iverson. **web Services** – Uma Solução para Aplicações Distribuídas na Internet

PETERSON, J. – **Notes on a Workshop on Distributing Computing** ACM Operating System Review, 13(3):18-30, jul. 1979.

QIU, Xiaohong, CARPENTER, Bryan e FOX, Geoffrey. Collaborative SVG as a web service.

QIU, Xiaohong, CARPENTER, Bryan e FOX, Geoffrey. Internet Collaboration using the W3C Documet Object Model.

QIU, Xiaohong, PALLICKARA, Shrideep e UYAR, Ahmet. Making SVG a web Service in a Message-based MVC Architecture.

QIU, Xiaohong. Message-based MVC Architecture for Distributed and Desktop Aplications. 2005. Dissertação (Doutorado em filosofia de ciência da computação) Universidade de Syracuse.

QIU, Xiaohong, PALLICKARA Shrideep e UYAR Ahmet. Making SVG a web Service in a Message-based MVC Archicterure.

QUARTO, Cícero. **Redes de Computadores e Internet. Aspectos Gerais**. CEFET – Maranhão

QUIN, Lean. **Extensible Markup Language (XML)**. Disponivel em: < http://www.w3.org/XML/ > Acesso em 11 de Abril de 2006.

SANTOS, Ricardo, MEIRELLES, Margareth, MAGALHÃES, Paulo. **Utilização da XML/GML, XML/XSLT e XML/SVG no Contexto das Funcionalidades Relacionadas às Estratégias de Visualização Cartográfica:** Uma Discussão Introdutória. Universidade do Estado do Rio de Janeiro.

SVG Docs., Scalable Vector Graphics. XML Graphics for the web. Disponível em: <http://www.w3.org/Graphics/SVG/>. Acesso em 19 fev. 2005.

SVG Profiles. Móbile SVG Profiles: SVG Tiny and SVG Basic. Disponível em: <http://www.w3.org/TR/SVGMobile/>. Acesso em 19 fev. 2005.

SVG. About SVG: 2d Graphics in XML. Disponível em: <http://www.w3.org/Graphics/SVG/About.html>. Acesso em 19 fev. 2005.

TANENBAUM, Andrew S. - Computer Networks – 1992.

TEMPLE, André; MELLO, Rodrigo Fernandes; GALEGARI, Danival Taffarel; SCHIEZARO, Mauricio. Programação web com com JSP, Servlet e J2EE.

Tutorial web Services. Disponível em: <http://www.sac.com.br/html/tutorial_web_services.html>. Acessado em 6 março 2005.

Tutorial web Services. Sistemas distribuídos. Disponível em: <http://twiki.im.ufba.br/bin/view/MAT167/TutwebServices>. Acessado em 7 março 2005.

WINBLAD, Ann L., EDWARDS, Samuel D., KING David R. **Object-Oriented Software** – 1990.