Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

**CELSO YAMAGUTI SOBRAL**

# TECNOLOGIAS JAVA PARA DESENVOLVIMENTO WEB UTILIZANDO A API GOOGLE MAPS

ASSIS
2010

Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

**CELSO YAMAGUTI SOBRAL**

# TECNOLOGIAS JAVA PARA DESENVOLVIMENTO WEB UTILIZANDO A API GOOGLE MAPS

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação.

Orientadora: Profa. Dra. Marisa Atsuko Nitto

Área de Concentração: Informática.

Assis
2010

## FICHA CATALOGRÁFICA

SOBRAL, Celso Yamaguti
Técnologias Java para desenvolvimento Web utilizando a API Google Maps / Celso Yamaguti Sobral. Fundação Educacional do Município de Assis – FEMA – Assis, 2010.
90p.

Orientadora: Profa. Dra. Marisa Atsuko Nitto.
Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1.Java. 2.Google Maps. 3.Web Service.

CDD:001.6
Biblioteca da FEMA.

Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

# TECNOLOGIAS JAVA PARA DESENVOLVIMENTO WEB UTILIZANDO A API GOOGLE MAPS

**CELSO YAMAGUTI SOBRAL**

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito de Curso de Bacharelado em Ciência da Computação, analisado pela seguinte comissão examinadora:

Orientadora: Profa. Dra. Marisa Atsuko Nitto

Analisador: Prof. Dr. Almir Rogério Camolesi

ASSIS
2010

# DEDICATÓRIA

*Dedico este trabalho, à minha querida mamãe Telma Maria Yamaguti, pelo seu grande incentivo, dedicação, compreensão e amor em todos os momentos desta e de outras caminhadas, ao meu pai José dos Santos Sobral (in memoriam), ao meu padrasto Cido, ao meu irmãozinho Danilo, à minha família e amigos.*

# AGRADECIMENTOS

Agradeço primeiramente a Deus, que me deu forças e sabedoria para concluir mais essa etapa da minha vida;

À minha família, principalmente à minha mãe, Telma, pelo amor, dedicação e confiança, estando presente em todos os momentos na caminhada sem medir esforços para a conclusão deste curso;

À minha orientadora, Marisa Atsuko Nitto, a quem devoto a mais sincera admiração, por toda sabedoria, apoio e incentivo para a conclusão deste trabalho e também por sempre compartilhar seu conhecimento e seu tempo durante todo o curso;

À Fundação Educacional do Município de Assis por ter-me possibilitado a realização deste trabalho;

Aos meus professores: Alex, Almir, Begosso, Célio, Dani, Diomara, Domingos, Fábio, Fernando, Guto, Leonor, Osmar, Regina, Rita e Talo por passarem todos os conhecimentos de informática;

Aos colegas de curso e aos grandes amigos pelo carinho e força que sempre me deram e por estarmos sempre juntos nos momentos mais importantes e por poder "contar" com vocês!;

E a todos aqueles que, direta ou indiretamente, colaboraram para que este trabalho consiga atingir aos objetivos propostos.

***Durante este trabalho...***

*As dificuldades não foram poucas...*
*Os desafios foram muitos...*
*Os obstáculos, muitas vezes, pareciam intransponíveis.*
*Muitas vezes me senti só, e, assim, estive...*
*O desânimo quis contagiar, porém, a garra e a tenacidade foram mais fortes, sobrepondo esse sentimento, fazendo-me seguir a caminhada, apesar da sinuosidade do caminho.*

*Agora, ao olhar para trás, a sensação do dever cumprido se faz presente e posso constatar que as noites de sono perdidas, as viagens e visitas realizadas; o cansaço dos encontros, os longos tempos de leitura, digitação, discussão; a ansiedade em querer fazer e a angústia de muitas vezes não o conseguir nas primeiras semanas de tentativa; não foram em vão.*

*Aqui estou, como sobrevivente de uma longa batalha, porém, muito mais forte e hábil, com coragem suficiente para mudar a minha postura, sonhar alto e alcançar todos os meus objetivos.*

*Como dizia Antoine de Saint Exupèry em sua obra prima "O Pequeno Príncipe":*

***"Foi o tempo que perdeste com a tua rosa, que fez a tua rosa tão importante."***

# RESUMO

O dinamismo e interatividade fazem parte de várias aplicações presentes na internet. Atualmente os usuários, além de usufruírem das informações, passaram também a utilizar vários serviços comerciais disponibilizados na web. Desta forma, uma tecnologia que tem sido bastante explorada é o *WebMap*, que possibilitam gerar mapas na internet com riqueza de informações espaciais e interatividade com os usuários.

Grandes empresas que possuem um comércio eletrônico geralmente disponibilizam um *web service* para que seu sistema possa trocar dados com outros sistemas que estão na *internet*. O uso desta tecnologia reduz a heterogeneidade e visa facilitar a integração de aplicações.

Em se tratando de *web service* e de comércio eletrônico, a tecnologia Java se sobressai em relação às outras existentes. A linguagem de programação Java apresenta uma série de características que a torna muito atrativa para o desenvolvimento de aplicações. Essas características abrangem a orientação a objetos, independência da plataforma, robustez, segurança, confiabilidade e conectividade com a Web. Isto proporciona o desenvolvimento de projetos com maior qualidade, portabilidade e com menor tempo de desenvolvimento.

Este trabalho apresenta a especificação e a implementação de um aplicativo para o setor imobiliário, desenvolvido em JSF (*Java Server Faces*) e utilizando a API do *Google Maps,* que é uma interface de desenvolvimento baseadas no próprio *Google Maps*, que ajuda a integrar mapas e geo-codificação aos sites. Isto possibilita que as aplicações com conteúdo geo-referenciado sejam facilmente apresentadas em qualquer navegador.

**Palavras Chaves**: Java; Web Service; Google Maps.

# ABSTRACT

The dynamism and interactivity belongs of various applications present on the internet. Actually, users beyond to take advantage of the information, also started to use several commercial services available on the web. Thus, a technology that's been explored is the WebMap, which may generate maps on the Internet with a wealth of spatial information and interactivity with users.

Large companies that have an e-commerce, generally provide a web service, to their system exchange data with other systems on the internet. This technology reduces the heterogeneity and facilitate the integration of applications.

When it comes to web services and e-commerce, Java technology goes on on the other. The Java programming language presents a number of features that makes it very attractive for developing applications. These features include object orientation, platform independence, robustness, security, reliability and connectivity to the Web. This provides project development with higher quality, portability and less development time.

This paper presents the specification and implementation of an application for the real state agency, developed in JSF (Java Server Faces) and using the Google Maps API, which is an interface of development based on Google Maps, wich helps integrate maps and geo-coding to sites. This allows applications with geo-referenced content to be easily displayed on any browser.



**Keywords**; Java; Web Service; Google Maps.

# LISTA DE ILUSTRAÇÕES

# SUMÁRIO

# CAPITULO 1

# INTRODUÇÃO

Com o grande crescimento da *internet* e do *e-Commerce*, várias empresas que antes utilizavam sistemas que focavam apenas atender a organização internamente, mudaram sua percepção para acompanhar o ritmo da evolução tecnológica e começaram a prover serviços da sua empresa na *internet*. O *e-Commerce* ou comércio eletrônico é a combinação entre o negócio tradicional e a automatização trazida pela *Internet* que permite às empresas, trocar informações ou dados sobre vendas, realizar transações financeiras, entregar e faturar bens e serviços, de uma forma automatizada e sobre um protocolo de comunicação seguro (KORPER; ELLIS, 2000).

Grandes empresas que possuem um comércio eletrônico geralmente disponibilizam um serviço *web* para que seu sistema possa conversar com outros sistemas que estão na *internet*. Uma *Web Service* ou serviço *web* é uma maneira de expor as funcionalidades de um sistema de informações e tornando disponíveis por meio de tecnologias *Web* padrões (ALONSO et al., 2004) e (RODRIGUES, 2008). O uso destas tecnologias reduz a heterogeneidade e visa facilitar a integração de aplicações.

No desenvolvimento deste projeto será utilizado o *Java Server Faces* (JSF), que é uma tecnologia de elaboração de sites com conteúdos dinâmicos. A vantagem desta tecnologia é que ela apresenta suas principais características herdadas da própria linguagem Java como: portabilidade, facilidade de programação, flexibilidade, eficiência e também, não menos importante, o fato de ser uma tecnologia gratuita, o que permite ao desenvolvedor usufruir todos os recursos disponíveis (BONFIN JUNIOR, 2000) e (BARALE, 2007). Sem contar que no mundo Java surgiu diversos *frameworks* e API (*Application Programming Interface*) para auxiliar no desenvolvimento *web*.

Neste trabalho, foi utilizado a API do *Google Maps* que é uma interface de desenvolvimento para aplicações baseadas no próprio *Google Maps*, permitindo criar aplicativos inovadores de mapeamento para a plataforma da *Web*. Essa API ajuda a integrar mapas e geocodificação aos sites, possibilitando que as aplicações com conteúdo geo-referenciado sejam facilmente apresentadas em qualquer navegador. A API consiste basicamente de um pequeno conjunto de classes *JavaScript* que fornecem uma interface para que o usuário possa construir aplicações para exibir os mapas, realizar consultas por endereços, realizar funções de zoom, acrescentar pontos de referência ou descrições no mapa, dentre outras possibilidades.

Este trabalho visa fundamentalmente adquirir conhecimentos das tecnologias utilizadas na área *Web*, com o intuito de desenvolver aplicativos para o setor imobiliário.

## 1.1 – Objetivos

Este trabalho tem como objetivo principal desenvolver um aplicativo utilizando as tecnologias Java para desenvolvimento *Web*. As tecnologias envolvidas para o desenvolvimento do aplicativo são *Servlets*, JSP (*Java Server Pages*), JSF (*Java Server Faces*), servidor de aplicação Appach Tomcat, XML (*Extensible Markup Language*), AJAX e a API *Google Maps* para manipulação de dados espaciais. Inicialmente, foram adquiridos os conhecimentos necessários das tecnologias utilizadas. Com os conhecimentos adquiridos, foi desenvolvido um aplicativo *web* inovador de atendimento a clientes para o setor imobiliário, sendo capaz de apresentar os produtos e serviços disponibilizados, tirar dúvidas sobre procedimentos, apresentar os imóveis disponíveis em pauta de acordo com as características desejadas pelo cliente. Para melhorar a apresentação dos imóveis, serão integradas ao aplicativo operações para buscar os imóveis mais similares de acordo com as características informadas pelo cliente e a sua localização no *Google Maps*.

## 1.2 – Motivação

Uma das principais motivações para o desenvolvimento deste trabalho, sem dúvida alguma, é o crescente mercado de serviços *web*. O avanço tecnológico tem propiciado desenvolvimento de aplicativos cada vez mais sofisticados, e isso requer conhecimento cada vez maior das tecnologias envolvidas nesta área. Algumas das tecnologias que serão abordadas visam outro mercado futuramente, que é o de desenvolvimento de aplicativos para dispositivos móveis. Além disso, será apresentado um produto final de grande relevância e inovador para o mercado imobiliário.

## 1.3 – Justificativa

Java é uma linguagem poderosa em ambientes distribuídos complexos como a *Internet*. O que a torna tão atraente é o fato de programas escritos em Java poderem ser executados *virtual*mente em qualquer plataforma, mas principalmente em Windows, Unix e Mac. Sua versatilidade permite ao programador ir além, oferecendo uma poderosa linguagem de programação de uso geral, com recursos suficientes para a construção de uma variedade de aplicativos que podem ou não depender do uso de recursos de conectividade (BUHR; VITORASSO, 2004), (BROGDEN; MINNICK, 2002) e (DEITEL; DEITEL, 2005). Sendo assim, atualmente o mercado de trabalho requer uma grande demanda de profissionais com conhecimento dessa tecnologia.

Outro fator importante foi o estudo da API do *Google Maps*, que está sendo cada vez mais utilizada e aprimorada pela Google, fazendo com que o conteúdo dos sites seja mais dinâmico e de fácil acesso com a utilização de mapas. Esses fatos por si só já justificam o desenvolvimento deste trabalho.

## 1.4 – Estrutura do trabalho

Este trabalho está subdividido em capítulos que serão explicitados a seguir.

O primeiro capítulo apresenta a contextualização e justificativa para o desenvolvimento da proposta do trabalho.

O segundo capítulo aborda a fundamentação teórica dos conceitos básicos que foram utilizados para o desenvolvimento e modelagem do projeto.

O terceiro capítulo apresenta o desenvolvimento do trabalho, mostrando a modelagem do problema, os diagramas de classe, casos de uso e diagramas de seqüência. Será feita uma descrição da implementação do aplicativo e os resultados obtidos.

O quarto capítulo apresenta a conclusão do trabalho.

# CAPITULO 2

# FUNDAMENTAÇÃO TEÓRICA BÁSICA

Neste capítulo, será feita uma fundamentação teórica dos conceitos básicos que foram utilizados para o desenvolvimento e modelagem do projeto. Estes conceitos se referem às linguagens e as ferramentas necessárias para o desenvolvimento do aplicativo.

## 2.1 – Linguagem Java

O berço da linguagem Java é um projeto da *Sun Microsystems* de 1991, chamado de Projeto Green, que tinha como mentores Patrick Naughton, Mike Sheridan, e James Gosling. Era um projeto sobre *software* para produtos eletrônicos de consumo. O objetivo era que os eletrônicos tivessem um *software* embarcado que possibilitasse o seu controle via rede de computadores e uma maior interatividade com o usuário (DEITEL; DEITEL, 2005), (SILVEIRA; COSENTINO, 2009) e (DRJT, 2010).

A linguagem de programação utilizada nestes aparelhos deveria proporcionar uma série de benefícios, entre eles a portabilidade, ou seja, um mesmo programa deveria poder ser executado em várias plataformas diferentes de *hardware*, a eficiência e a perspectiva de vida longa, o que, em outras palavras, pode significar compatibilidade retroativa, ou melhor, os velhos programas deveriam funcionar em plataformas novas.

A tecnologia Java foi projetada para se mover através de redes de dispositivos heterogêneos, como a *Internet*. Agora aplicações poderiam ser executadas dentro dos *Browsers* nos *Applets* Java e tudo seria disponibilizado pela *Internet* instantaneamente. Foi o estático HTML dos Browsers que promoveu a rápida disseminação da dinâmica tecnologia Java. A velocidade dos acontecimentos

seguintes foi assustadora, o número de usuários cresceu rapidamente, grandes *players*, como a IBM anunciaram suporte para a tecnologia Java.

### 2.1.1 - O que é Java

Java não é somente uma linguagem, como também é uma tecnologia. Basicamente, ela se constitui de uma linguagem de programação e um programa para execução chamado de máquina virtual ou *virtual machine*. Quando programa-se em Java usa-se a linguagem de programação e um ambiente de desenvolvimento para gerar um *software* que será executado em um ambiente de distribuição.

Java é uma linguagem que não se prende a nenhuma arquitetura e a nenhuma empresa, é rápida e estável. Pode construir sistemas críticos, sistemas que precisam de velocidade e até sistemas que vão para fora do planeta, como a sonda *Spirit* enviada pela NASA (*National Aeronautics and Space Administration*) para Marte. Java tem um mar de projetos *open source*, que estão lá, esperando por usuários e desenvolvedores (DEITEL; DEITEL, 2005), (JAVAFREE, 2010) e (SILVEIRA; COSENTINO, 2009).

### 2.1.2 - Principais Características da Linguagem Java

As principais características da linguagem Java serão descritas para facilitar o entendimento da programação.

- ***Write once, run everywhere***: escreva uma vez e execute em qualquer lugar. Este é o motor de Java, o que impulsionou a linguagem, a característica de compatibilidade do código Java com qualquer *hardware*. A promessa que navegadores *web*, telefones, PDAs e outros aparelhos eletrônicos pudessem executar um mesmo código fonte Java ajudou a popularizar a linguagem e atrair novos programadores;

- ***Web programming*:** a Sun fez um prognóstico arrojado para a época afirmando que a *web* era um computador logo, deveriam existir ferramentas para "programar" este novo dispositivo, a rede. A arquitetura cliente/servidor é algo quase que natural em Java;
- **Recursos de Rede**: possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP;
- **Segurança**: permite que usuários utilizem *software*s seguros que foram copiados da *web*;
- **Programação dinâmica e extensível**: a programação Java é baseada integralmente em POO (Programação Orientada á Objetos), ou seja, é baseada na descrição de classes. Para conceitos novos e tarefas novas criam-se novas classes ou, realizam-se a manutenção ou incrementam-se classes antigas;
- **Eficiência na codificação**: com a utilização de classes, a programação torna-se mais eficiente já que as chances de reescrever *software* inédito tornam-se menor, ou seja, sempre existirá uma classe que faz aquilo que o programador deveria implementar;
- **Internacionalização**: é o poder de escrever código fonte que trabalhe com caracteres dos mais diferentes alfabetos. Este foi um dos últimos recursos, dos citados aqui, a ser implementado em Java;
- ***Garbage collector***: objetos não utilizados são retirados da memória principal automaticamente. Isto se chama deslocamento automático de memória dinâmica;
- **Concorrência**: permite o uso de threads múltiplos e mecanismos explícitos de concorrência;
- **Tratamento de eventos e exceções**: permite a programação orientada por eventos e a possibilidade de escrever programas que respondem a falhas de execução;
- **Carga Dinâmica de Código**: programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

### 2.1.3 - Máquina Virtual Java

As linguagens de programação, em sua maioria, ou são compiladas ou interpretadas. Com Java acontecem as duas transformações: o código fonte Java é inicialmente compilado produzindo um código intermediário chamado Java *bytecode* conhecido por arquivos com a terminação .class; só então um interpretador faz um *parse* (tradução) e executa o Java *bytecode* (DEITEL; DEITEL, 2005) e (SILVEIRA; COSENTINO, 2009). A figura 1, mostra como é o funcionamento interno de uma Máquina *Virtual* Java.

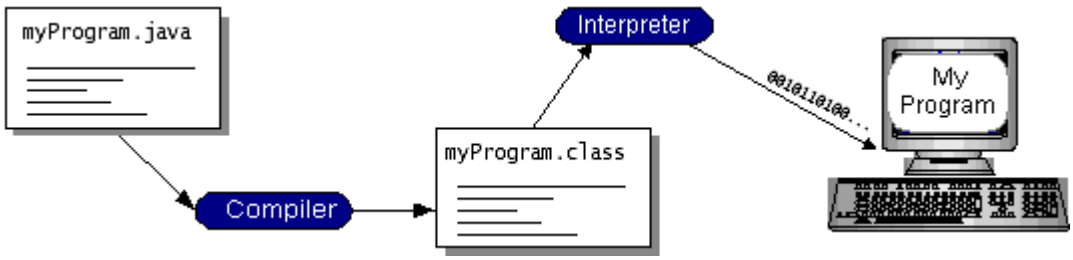


**Figura 1 - Funcionamento interno da JVM. Fonte: (RUIZ, 2010).**

Programas Java não são traduzidos para a linguagem de máquina como outras linguagens estaticamente compiladas e sim para uma representação intermediária, chamada de *bytecode*s. Os *bytecode*s são interpretados pela máquina virtual Java (JVM - Java *Virtual Machine*). Muitas pessoas acreditam que por causa desse processo, o código interpretado Java tem baixo desempenho. Durante muito tempo esta foi uma afirmação verdadeira. Porém novos avanços tem tornado o compilador dinâmico (a JVM), em muitos casos, mais eficiente que o compilador estático.

Todo interpretador Java, independentemente da plataforma ou do *software* em que se insere, é conhecido como uma Java *Virtual Machine* (JVM). A figura 2 ilustra este processo de compilação e interpretação.

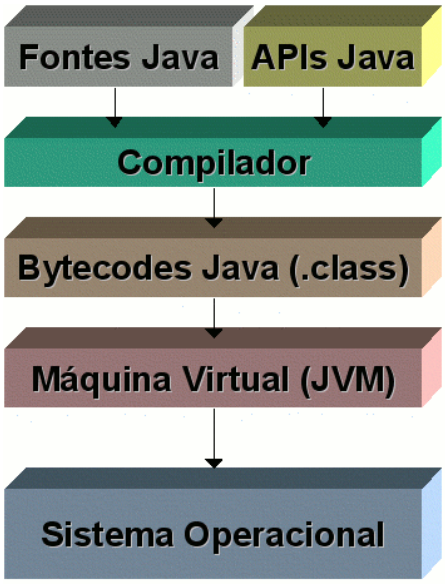


**Figura 2 - Processo de compilação Java. Fonte: (RUIZ, 2010).**

Isto garante que todo *hardware* que executa uma JVM pode executar um mesmo código fonte escrito em Java. A figura 3 mostra esta execução.

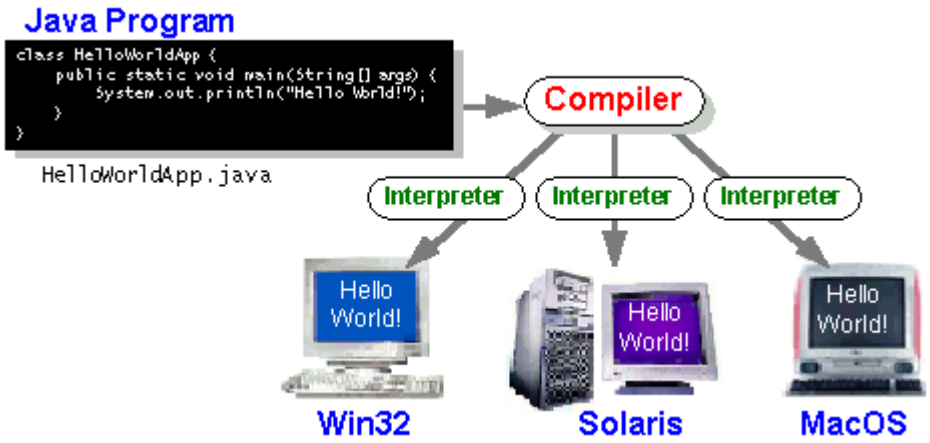


**Figura 3 - Execução do programa Java em diversas plataformas. Fonte: (RUIZ, 2010).**

Atualmente, Java já possuiu uma *performace* próxima do C++. Isto é possível graças às otimizações como a compilação especulativa, que aproveita o tempo ocioso do processador para pré-compilar *bytecode* para código nativo. Outros mecanismos

ainda mais elaborados como o *HotSpot* da *Sun*, que guarda informações disponíveis somente em tempo de execução (ex.: número de usuários, processamento usado, memória disponível), para otimizar o funcionamento da JVM, possibilitando que a JVM vá "aprendendo" e melhorando seu desempenho. Isto é uma realidade tão presente que hoje é fácil encontrar programas corporativos e de missão crítica usando tecnologia Java. No Brasil, por exemplo, a maioria dos Bancos utiliza a tecnologia Java para construir seus *home banks*, que são acessados por milhares de usuários diariamente. Grandes sites como o *eBay* utilizam Java para garantir alta *performace*.

### 2.1.4 - Plataforma

Uma plataforma é o *hardware* ou o *software* em que um sistema ou um programa é executado. Atualmente este termo é mais utilizado para o conjunto de *hardware* e *software*. No entanto, a realidade da plataforma Java é um pouco diferente, pois é entendida como uma plataforma só de *software*, composta pela JVM e pela Java API. A Java API é um conjunto de componentes de *software* prontos para serem usados. Em Java existem APIs para uma variedade de aplicações, desde tratamento de recursos gráficos, a métodos para comunicação em redes e funções matemáticas.

### 2.1.5 - Edições Java

A expansão das bibliotecas padrão do Java, e o “engordamento” da JVM focada em PCs, levou à criação do Java2, sinalizando a divisão da plataforma Java em três ou “*profiles*” distintos (LOZANO, 2010). Será feita uma descrição detalhada das três edições Java.

#### 2.1.5.1 - *Java 2 Standard Edition* (J2SE)

É a tecnologia Java para computadores pessoais, *notebooks* e arquiteturas com poder de processamento e memória consideráveis. Várias APIs acompanham esta versão e tantas outras podem ser baixadas opcionalmente no site da Sun. Com elas que a maioria das aplicações é construída e executada. A figura 4, mostra a plataforma do Java 2 *Platform Standard Edition*. O J2SE possui duas divisões:

- ***Java Development Kit* (JDK) ou *Standard Development Kit* (SDK)**: um conjunto para desenvolvimento em Java e deveria ser instalado apenas pelos desenvolvedores por possuir ferramentas para tal tarefa;

- ***Java Runtime Edition* (JRE)**: uma versão mais leve da JDK, pois é preparada para o ambiente de execução, ou seja, é esta versão que executará os sistemas construídos com a SDK.

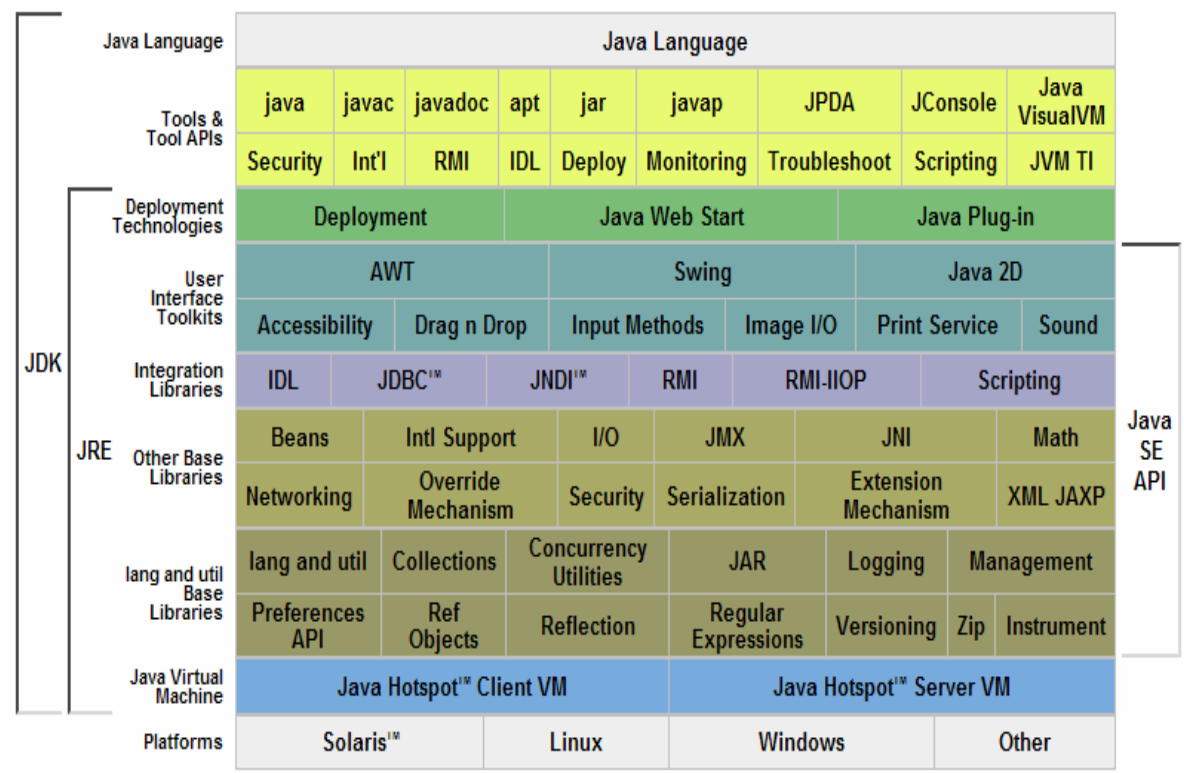


**Figura 4 - Plataforma do Java 2 Standard Edition (J2SE). Fonte: (DRJT, 2010).**

### 2.1.5.2 - *Java 2 Mobile Edition* (J2ME)

É a tecnologia Java para dispositivos móveis com limitações de memória ou processamento. Possui APIs bem simples e leves para economizar espaço, memória e processamento. São utilizadas para sistemas em celulares*, palm tops, pocket pcs*, *smartphones*, *javacards* e demais dispositivos. A figura 5, mostra a Plataforma *Java 2 Mobile Edition*.

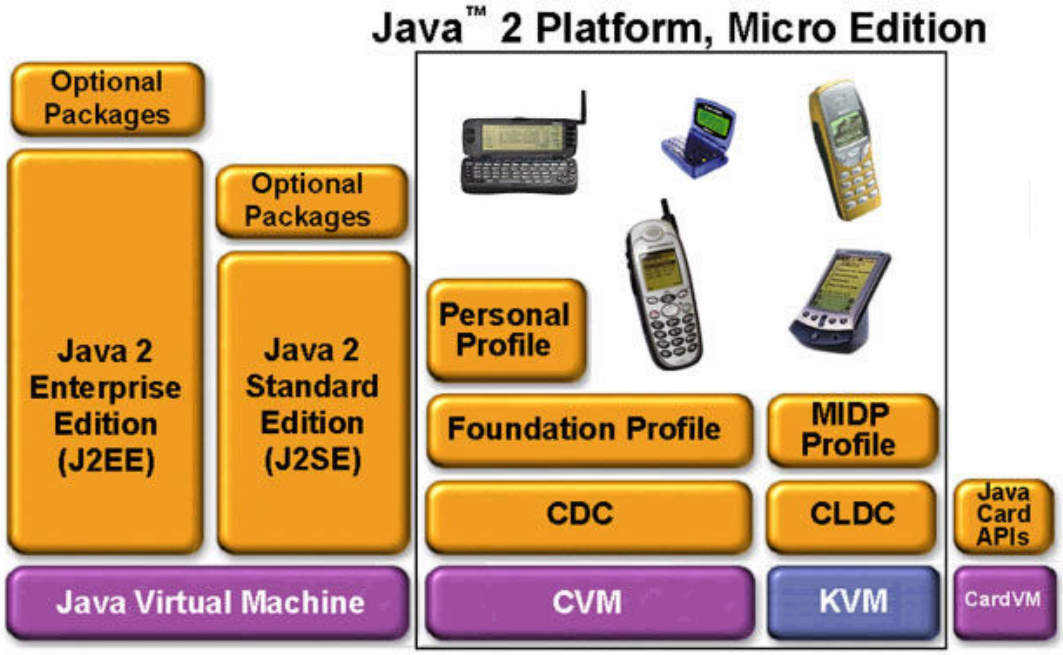


**Figura 5 - Plataforma Java 2 Mobile Edition. Fonte: (ALVES, 2010).**

O J2ME se divide em dois grupos de bibliotecas e que serão descritas para melhor entendimento:

- ***Connected Limited Device Configuration*** **(CLDC):** para celulares e smartphones, que são mais limitados;

- ***Connected Device Configuration*** **(CDC):** para Palmtops e Pocket pcs e alguns dispositivos mais poderosos.

### 2.1.5.3 - *Java 2 Enterprise Edition* (J2EE)

É a tecnologia Java para aplicações corporativas que podem estar na *internet* ou não. Possui um grande número de APIs onde a segurança é a principal preocupação. É ideal para a construção de servidores de aplicação, integração de sistemas ou distribuição de serviços para terceiros. Ela apresenta facilidades para a

utilização dos recursos computacionais e distribuídos tais como acesso à banco de dados, componentes *Web*, utilização de mensagens assíncronas, execução de processos transacionais, persistentes ou não. Apresenta uma API, especificada pela *Sun MicroSystems*, que proporciona um padrão para a implementação dos diversos serviços que oferece, sendo que isto pode ser feito diferentemente por várias empresas, de formas distintas mas ainda assim oferecendo as mesmas facilidades, por estarem de acordo com as especificações impostas para a sua construção (ARMSTRONG et al., 2005) e (FISHER; GREEN, 2010).

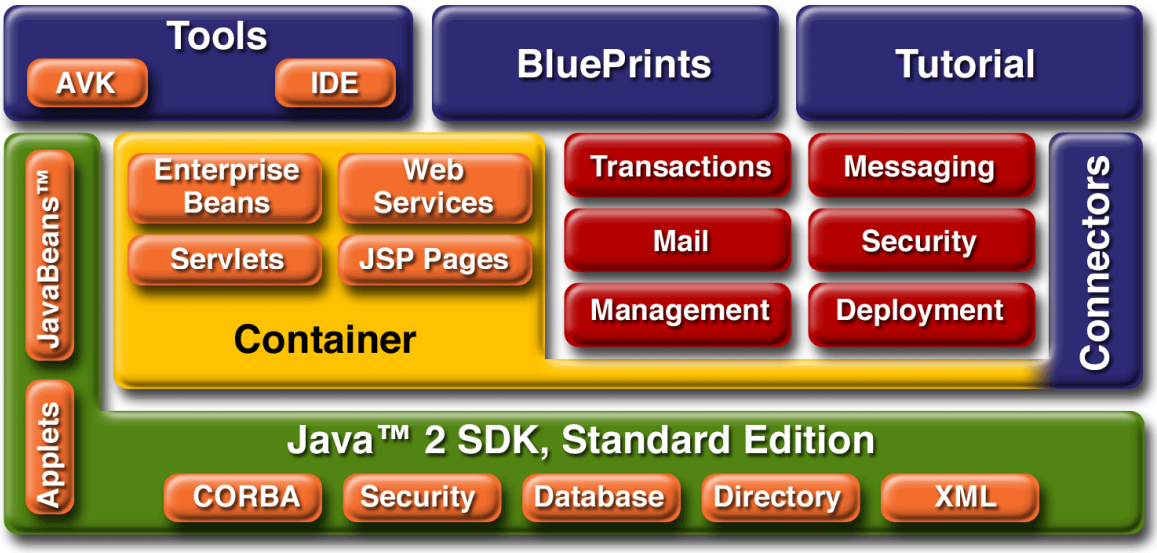


**Figura 6 - Plataforma Java 2 *Enterprise Edition*. Fonte: (GRANADOS, 2010).**

A arquitetura J2EE se apresenta em várias camadas, sendo que cada camada é composta por componentes e serviços que são providos por um *container*. Em um servidor J2EE, existem diversos *containers* interagindo entre si. Será feita uma descrição detalhada de cada camada da arquitetura e de seus componentes:

- **Camada cliente**: acesso por meio de interfaces *stand-alone* (aplicações Java), páginas HTML ou *Applets*. Nesta camada os componentes residem em um *Applet Container*, em um HTML *container* (*Web browser*) ou em um *Application Client Container*. O *Applet container* é aquele que fornece os recursos para um

componente *Applet* executar e se tornar funcional para o usuário. A *Web Browser* apresenta recursos e funcionalidades para o uso de páginas HTML e por fim o *Application Client container* fornece recursos para a execução das classes *stand-alone* utilizadas pelos usuários para interagirem no sistema;

- **Camada *Web***: esta camada é implementada por JSPs e *Servlets* que fornecem a lógica para a camada cliente (ou de apresentação) do negócio. As JSPs e *Servlets* residem na *Web Container*. A JSPs oferece a facilidade de utilizar algumas lógicas de apresentação em uma página *web* sem muitas dificuldades tecnológicas. O *Servlet* apresenta-se como um controlador das ações executadas pelos usuários nas páginas de apresentação, e fornece recursos para obter dados dessas ações e realizar as operações desejadas. Os componentes *Web* residem na *Web* Container que pode ser um servidor *TomCat* ou outro similar.
- **Camada de Negócios**: esta camada trata da lógica de negócio da aplicação. É nela que se implementa todas as regras de negócio, alocação de recursos, persistência de dados, validação de dados, gerencia de transações e segurança, providos por componentes conhecidos por EJBs. Estes por sua vez residem no EJB Container.
- **Camada EIS** (*Enterprise Information System*): nesta camada é que se encontram os sistemas de banco de dados, sistemas legados, integração com outros sistemas não J2EE etc.

As três edições compartilham uma definição comum da JVM, embora hajam implementações especializadas para o J2ME (LOZANO, 2010). Dentro da arquitetura, as camadas podem se relacionar com os *containers* dependendo da aplicação. A figura 7 mostra a interação entre as camadas e *containers*.

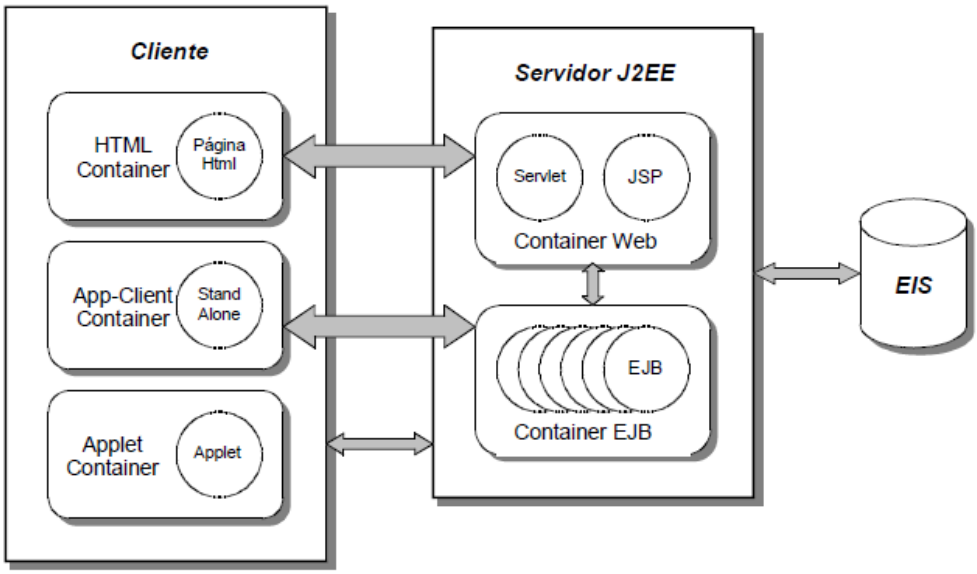


**Figura 7 - Interação entre camadas e *containers*. Fonte: (ARMSTRONG et al., 2005).**

A Plataforma Java Enterprise Edition difere-se da Plataforma Java Standard Edition pela adição de bibliotecas que fornecem funcionalidade para implementar *software* Java distribuído, tolerante a falhas e multi camada, baseada amplamente em componentes modulares executando em um servidores de aplicações. A plataforma Java EE é considerada um padrão de desenvolvimento já que o fornecedor de *software* nesta plataforma deve seguir determinadas regras se quiser declarar os seus produtos como compatíveis com *Java EE*. Ela contém bibliotecas desenvolvidas para o acesso a base de dados, RPC (*Remote Procedure Call*), CORBA (*Common Object Request Broker Architeture*), etc. Devido a essas características a plataforma é utilizada principalmente para o desenvolvimento de aplicações corporativas. Serão apresentadas as principais APIs do J2EE:

- **JSP (*JavaServer Pages*)**: uma especialização do *servlet* que permite que conteúdo dinâmico seja facilmente desenvolvido;

- **JS (*Java Servlets*)**: são utilizados para o desenvolvimento de aplicações *Web* com conteúdo dinâmico. Ele contém uma API que abstrai e disponibiliza os recursos do servidor *Web* de maneira simplificada para o programador;
- **EJB (*Enterprise Javabeans Components*)**: utilizados no desenvolvimento de componentes de *software*. Eles permitem que o programador se concentre nas necessidades do negócio do cliente, enquanto questões de infra-estrutura, segurança, disponibilidade e escalabilidade são responsabilidade do servidor de aplicações;
- **JPA (*Java Persistence* API)**: é uma API que padroniza o acesso a banco de dados através de mapeamento Objeto/Relacional do *Enterprise Java Beans*;
- **JTA (*Java Transaction* API)**: é uma API que padroniza o tratamento de transações dentro de uma aplicação Java;
- **JAAS (*Java Autenthication and Authorization Service*)**: é uma versão Java da infra-estrutura padrão PAM (*Pluggable Authentication Module*), que estende a arquitetura de segurança da plataforma Java para suportar autorização baseada em usuário;
- **JSF (*Java Server Faces*):** é um framework de aplicação *web* baseado em Java, destinada para simplificar a integração do desenvolvimento de interfaces de usuário baseadas na *web*;
- **JAX-WS (Java API *for* XML *Web Services*):** é uma tecnologia fundamental para o desenvolvimento de *WebServices* baseada em SOAP (*Simple Object Access Protocol*);
- **JAX-B (Java API *for* XML *Binding*):** fornece uma maneira conveniente para processar conteúdo XML usando objetos Java, ligando seu *Schema* de XML para representação em Java;
- **JMS (*Java Message Service*):** é uma API da linguagem Java para *middleware* orientado à mensagens. Através da API JMS duas ou mais aplicações podem se comunicar por mensagens;

- **JNDI (*Java Naming and Directory Interface*):** é parte da plataforma Java, fornecendo aplicações baseadas na tecnologia Java com uma interface unificada para vários nomes e serviços de diretório.

Para o desenvolvimento do aplicativo foram utilizadas algumas das API apresentadas, e para isso será feita uma descrição mais detalhada para melhor entendimento teórico.

### 2.1.6 - *Enterprise JavaBeans*

O *Enterprise JavaBeans* são objetos distribuídos que apresentam uma estrutura bem definida, isto é, implementam interfaces específicas e que executam no lado do servidor. Ela nada mais é do que simples objetos que devem seguir algumas regras. Estas regras foram definidas pela *Sun MicroSystems* através da especificação de EJBs na arquitetura J2EE. Apresentam métodos para a lógica de negócio e métodos que tratam da criação (instanciação), remoção, atualização do EJB entre outros, dentro do ambiente aonde sobrevive. Os EJBs não são simples classes Java, mas sim componentes distribuídos que fornecem serviços e persistência de dados, além de processamento assíncrono e que podem ser invocados remotamente. Eles são normalmente utilizados para executarem a lógica de negócio do lado do servidor de forma distribuída, sobrevivendo em ambientes distintos, em máquinas diferentes, em locais geograficamente diversos e ainda assim utilizando de serviços eficientes.

A vantagem na utilização dos EJBs é que a aplicação irá se beneficiar de serviços como transações, segurança, tolerância a falhas, *clustering*, distribuição, controle de sessão entre outros. Estes serviços são fornecidos pelo ambiente que o EJB sobrevive que é o container EJB. A figura 8 mostra um exemplo de acesso remoto a um serviço, utilizando a API de *Sockets* e o recurso de serialização de objetos realizado pelo *container*.

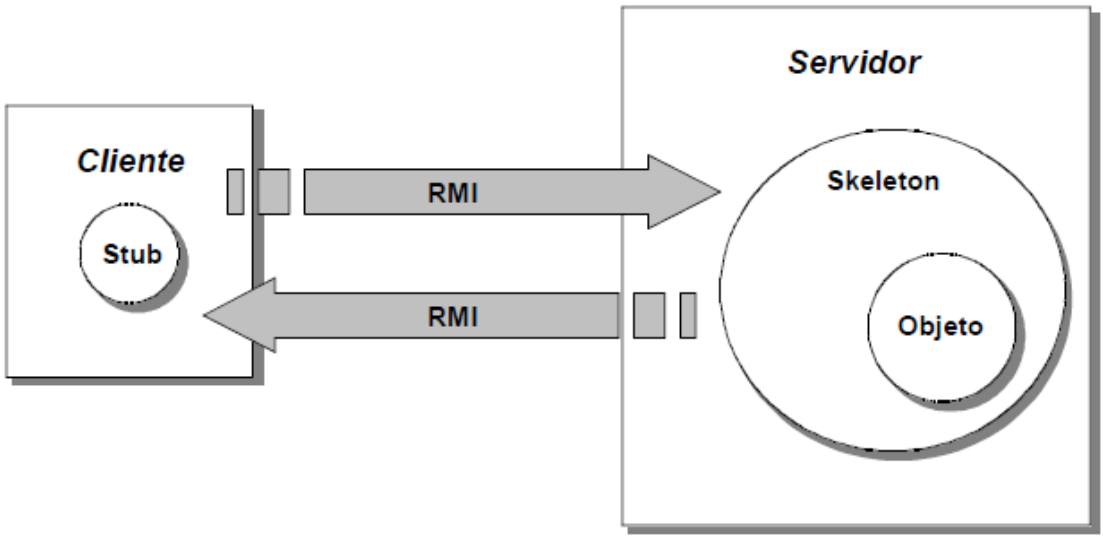


**Figura 8 - Exemplo de serviço e troca de objetos utilizando *Sockets* e Serialização. Fonte: (ARMSTRONG et al., 2005).**

### 2.1.7 - *Servlets*

Quando você digita um endereço no seu navegador para entrar em alguma página de *internet*, basicamente solicita um determinado arquivo localizado em um computador em especial no local que você digita a URL. O computador onde o arquivo está armazenado é chamado de *Web Server* (servidor *web*). Essa principal função do computador é para servir a qualquer um na *Internet* que solicite arquivos que ele se hospeda. Já que você nunca sabe quando um usuário visitará e usará o seu aplicativo *web*, o seu servidor *web* precisa estar ativo e em execução o tempo todo (ARMSTRONG et al., 2005), (FISHER; GREEN, 2010) e (SANCHEZ, 2004).

As ações que acontecem quando se digita um endereço na *internet* são:

- O *browser* (cliente) estabelece uma conexão TCP/IP com o servidor;
- O *browser* envia uma solicitação ao servidor (*request*);

- O servidor envia uma resposta ao cliente (*response*);
- O servidor fecha a conexão.

HTTP é o protocolo que permite que os servidores *web* e browsers troquem dados pela *web*. É um protocolo de solicitação e resposta.

No início as páginas eram praticamente todas estáticas, ou seja, somente arquivos HTML que não eram gerados dinamicamente e nem tinham conexões com banco de dados. Logo após começaram a criar as CGIs (*Common Gateway Interface*) que eram geradas dinamicamente, podiam trabalhar com arquivos, interagir com banco de dados e etc. Através de linguagens como C, C++, Perl, ASP e PHP foi possível gerar conteúdo que permite ao usuário acesso a diversas funcionalidades através de páginas HTML, como quando se deseja comprar produtos em uma loja virtual.

Para melhorar o desempenho da geração de conteúdo dinâmico, a Sun criou a especificação do que vem a ser uma *Servlet*, uma nova forma de trabalhar com requisições de clientes via *web* que economiza o tempo de criar e alocar memória para um processo no sistema operacional. *Servlet* é um programa que estende a funcionalidade de uma *Web Server*, gerando conteúdo dinâmico e interagindo com os clientes, utilizando o modelo *request/response*. Os *Servlets* não são restritos ao modelo HTTP de *request/response*, onde na realidade são pequenas aplicações de servidores, mas o modelo HTTP é o modelo mais comumente utilizado. Além do mais, *Servlets* são tão portáveis quanto qualquer programa escrito em Java, e aqueles que programam *Servlets* não precisam mais se preocupar com a funcionalidade do servidor.

Um cenário típico de funcionamento de uma aplicação desenvolvida em *Servlets* é mostrado na figura 9.

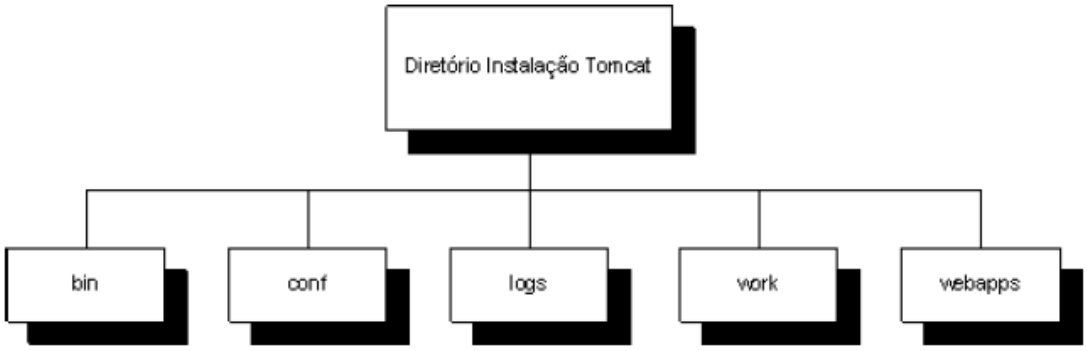


**Figura 9 - Cenário de Aplicação feita em *Servlet* . Fonte: (SANCHEZ, 2004).**

### 2.1.7.1 - Arquitetura de um *Servlet*

Todos os *servlets* implementam direta ou indiretamente a interface *Servlet*. O mais comum é o *servlet* dar *extends* na *HttpServlet* (que implementa a interface *Servlet*). A interface *Servlet* fornece métodos para gerenciamento do *servlet* e sua comunicação com clientes. Os protocolos existentes são:

- **Protocolo HTTP**

Embora *Servlets* possam ser utilizados não só para o desenvolvimento de aplicações HTTP, a maior parte das aplicações desenvolvidas é destinada a esse fim. Sendo assim, vale à pena estudar um pouco mais a fundo o funcionamento e características desse protocolo. O protocolo HTTP é utilizado na navegação nas páginas da *Internet* quando se abre uma janela de um browser, acessa uma página *Web* e navega em seus *links*. Na verdade, este protocolo é utilizando para visualizar, em sua máquina, o conteúdo que está armazenado em servidores remotos.

O HTTP é um protocolo *stateless* de comunicação cliente-servidor, isto é, o cliente envia uma requisição para o servidor, este processa a requisição e devolve uma resposta para o cliente, sendo que, a princípio, nenhuma informação é mantida no servidor em relação às requisições previamente recebidas. Assim, ao digitar o

endereço de uma página em um *browser Web*, será gerada uma requisição a um servidor, que irá, por sua vez, devolver para o *browser* o conteúdo da página HTML requisitada. A requisição enviada por um cliente deve conter, basicamente, um comando (também chamado de método), o endereço de um recurso no servidor (também chamado de "*path*") e uma informação sobre a versão do protocolo HTTP sendo utilizado.

- **HTTP *Servlet***

No Java existe a classe HTTP *Servlet*, na qual estende a classe *javax.servlet.GenericServlet*, que possui basicamente seis métodos que são chamados automaticamente de acordo com os métodos HTTP que são requisitados. Os seis métodos são:

- ***doPost*():** utilizado para envio de dados ao servidor, uma única vez;
- ***doGet*():** utilizado para envio de dados ao servidor, repetidas vezes;
- ***doPut*():** permite enviar um arquivo ao servidor;
- ***doDelete*():** permite remover um documento ou uma página do servidor;
- ***doOption*():** determina quais opções do HTTP são suportadas;
- ***doTrace*():** fornece resposta com todos cabeçalhos enviados.

Basicamente os métodos mais utilizados são: *doGet*() e *doPost*(). A figura 10 mostra um HTTP *Servlet*.

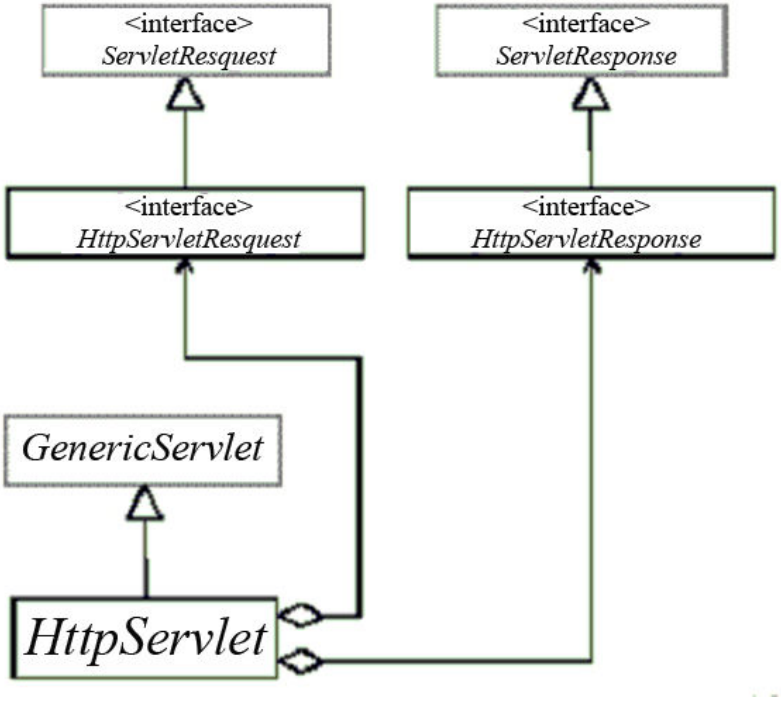


**Figura 10 - HTTP *Servlet*. Fonte: (SANCHEZ, 2004).**

- **HTTP *Servlet* Request**

As solicitações HTTP que o browser envia pelo cliente ao servidor com informações importantes, tais como *cookies* e referenciador são tratadas a partir do objeto *HttpServletRequest* passado a um método *doGet* ou *doPost*.

| **Método** | **Descrição** |
|---|---|
| *getHeaderNames*(); | pega todos os nomes dos cabeçalhos . |
| *getHeader*(); | pega todos os valores do cabeçalho . |
| *getQueryString*(); | pega a Query String completa. |
| *getParameterNames*(); | pega todos os names dos parâmetros passados. |

| | |
|---|---|
| *getParameterValues*(); | recuperação de parâmetros de múltiplos valores. |
| *getParameter*(); | recuperação de parâmetros de acordo com o nome passado. |

**Tabela 1 - Principais métodos do HTTP *Servlet* Request. Fonte: (SANCHEZ, 2004).**

- **HTTP *Servlet Response***

A interface *HttpServletResponse* oferece diversos métodos específicos de protocolo. Para isso, é necessário especificar a saída para seu browser, através dos métodos *setContentType* e *getWriter*. Além disso, é possível enviar cookies ao browser, também tem métodos para manipular os URLs enviados ao browser, gerenciamento de sessões, etc.

- ***Servlet Container***

O *Servlet Container* é um servidor de aplicações para executar *Servlets*. Ele é responsável por controlar o ciclo de vida do *Servlet* e interagir com a plataforma de execução. Quando a solicitação de execução do *Servlet* é feita ao *container*, caso não exista nenhuma instância do *Servlet* solicitado carregada no *container*, o *container* carrega a classe, cria uma instância e a inicia chamando o método *init*(), executa o método de serviço solicitado passando como parâmetro os objetos *request* e *responsee* quando ele não precisar mais do *servlet* ele executa o método *destroy*(). O *Servlet* container, ou servidor de aplicação, mais conhecido é o *Apache Tomcat*.

### 2.1.7.2 - Ciclo de vida

O ciclo de vida de um *servlet* é determinado por três fases: inicialização, atendimento de requisições e finalização, que correspondem respectivamente aos três métodos: init, *service* e *destroy*. Os servidores carregam e executam o *servlet* através de um *Servlet* Container, que por sua vez aceita nenhuma ou mais requisições de clientes e retorna dados para o mesmo. Os servidores podem também remover os *servlets* (ARMSTRONG et al., 2005) e (FISHER; GREEN, 2010). A figura 11 mostra o ciclo de vida do *Servlets*.

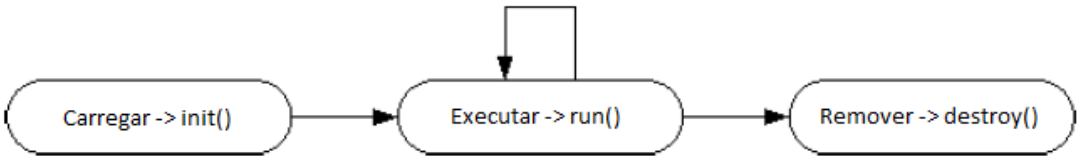


**Figura 11 - Ciclo de vida do *Servlet*. Fonte: (SANCHEZ, 2004).**

Os três pontos do ciclo vital do *servlet* são:

- **Carregar**: quando um servidor carrega um *servlet*, ele executa o método init() do mesmo. Como a maioria dos *servlets* executa em servidores *multi-thread*, não há concorrência durante a inicialização dos mesmos. Isto ocorre, pois o servidor chamará somente uma vez o método *init*() quando carregar o *servlet* e não o chamará de novo a não ser que o *servlet* seja recarregado. O servidor não pode recarregar um *servlet* até que o mesmo seja removido pelo método *destroy*();
- **Executar**: após o servidor carregar e incializar o *servlet*, o mesmo está apto a receber requisições do cliente. Cada requisição do cliente possui uma chamada ao método *run*() em sua própria thread do *servlet*. O método recebe a requisição do cliente e envia a resposta. Os *servlets* podem executar vários métodos de serviços de uma vez;

- **Remover**: Os *servlets* permanecem em execução até que sejam removidos. Quando um servidor remove um *servlet*, ele executa o método *destroy*() do mesmo. O método é executado somente uma vez. O servidor não executará o método destroy() novamente até que o *servlet* seja recarregado e reinicializado.

A figura 12 mostra o funcionamento de um Servelets.

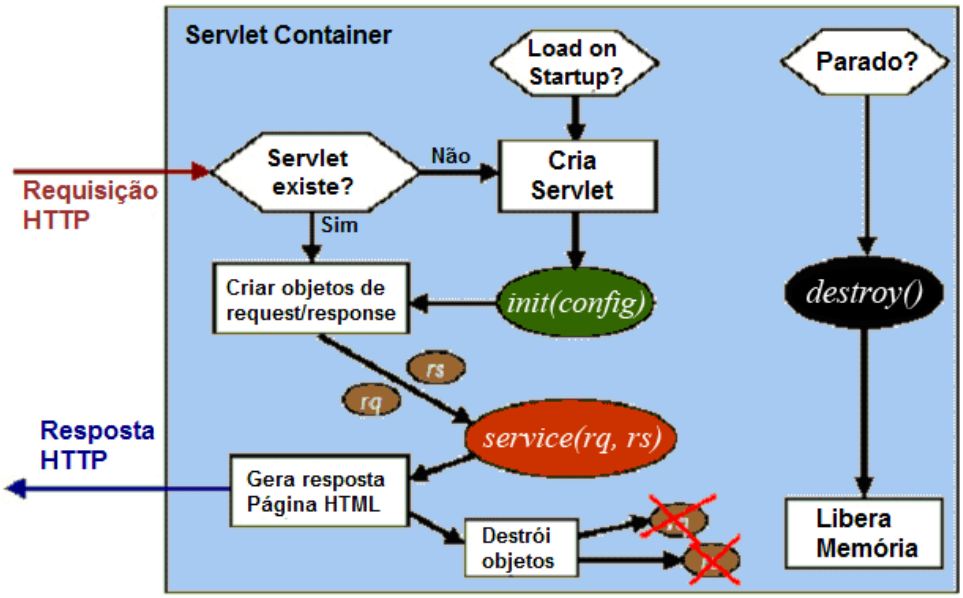


**Figura 12 - Funcionamento *Servlet*. Fonte: (SANCHEZ, 2004).**

### 2.1.7.3 - Contexto do *Servlet*

Em programação de *servlet*, o contexto de *servlet* é o ambiente onde o *servlet* executa. O container de *servlet* cria um objeto *Servlet*Context que será utilizado para acessar informações a respeito do ambiente do *servlet*. Um *servlet* também pode ligar um atributo de objeto ao contexto pelo nome. Qualquer objeto ligado a um contexto está disponível a qualquer *servlet* que faça parte do mesmo aplicativo *web*.

### 2.1.7.4 - Gerenciamento de sessão

O protocolo de rede que os servidores *web* e browsers cliente usam para se comunicar é o HTTP, sendo a linguagem *web*. As conexões HTTP são iniciadas por um browser cliente que envia uma solicitação HTTP. Então, o servidor *web* responde com uma resposta HTTP, num comportamento simplificado. Portanto, se um cliente solicitar outro recurso do servidor, será necessário abrir outra conexão HTTP para o servidor. O gerenciamento de sessão possui basicamente quatro técnicas que são:

- Reescrita de URL;
- Campos ocultos;
- *Cookies*;
- Objetos de sessão.

Dessas quatro técnicas de gerenciamento de sessão, o objeto *Session* representado pela *interface javax.servlet.http.HttpSession* é o mais fácil de usar e o mais poderoso. Por padrão, quando um objeto *Session* é criado ele envia ao browser do cliente e ao servidor um identificador de sessão, que fica ativo até que se destrua a sessão ou expire o tempo. A figura 13 mostra como é realizado o gerenciamento de sessão.

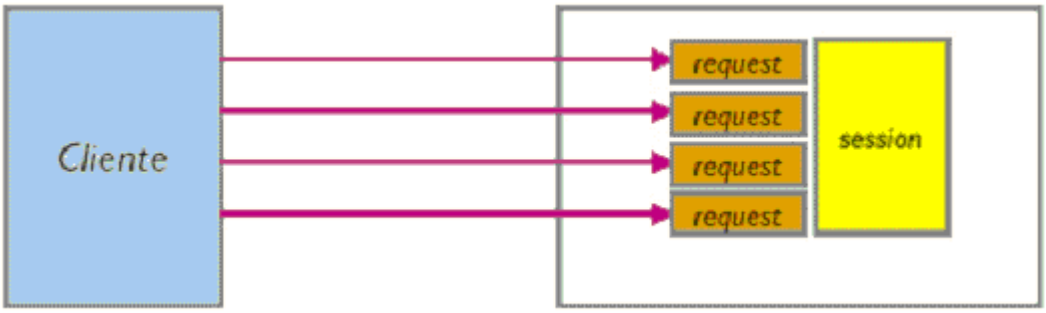


**Figura 13 - Gerenciamento de Sessão. Fonte: (SANCHEZ, 2004).**

Os principais comandos do objeto Session são:

- **Criar Sessão**: *HttpSession session = request.getSession( true );*
- **Recuperar Sessão**: *request.getSession();*
- **Atribuir Valor**: *session.setAttribute("*usuario*","*valor*");*
- **Recuperar Valor**: *session.getAttribute("*usuario*");*
- **Destruir Sessão**: *session.invalidate();*

### 2.1.8 - *Java Server Pages* (JSP)

A Tecnologia JSP é um componente chave na plataforma *Java 2 Enterprise Edition*, que é uma arquitetura altamente escalonável da *Sun* para aplicações empresariais. Uma página JSP é uma página que possui uma estrutura fixa somada com algum tipo de linguagem de marcação para incluir outro tipo de texto ou lógica embarcada. Esta estrutura fixa normalmente é toda baseada em HTML. A linguagem de marcação que tem como função gerar algum tipo de conteúdo dinâmico pode apresentar-se das seguintes formas: scripts ou *tags* customizadas (SILVEIRA; COSENTINO, 2009) e (BONFIN JUNIOR, 2002). As páginas JSP partilham as características da tecnologia do Java "*Write Once, Run Anywhere*".

#### 2.1.8.1 - Aplicabilidade e Uso

A tecnologia JSP é parte da família de tecnologias Java. Páginas JSP são compiladas em *servlets* e podem chamar componentes *JavaBeans* (*beans*) ou componentes *Enterprise JavaBeans* (*enterprise beans*) para a realização do processamento no servidor. A tecnologia JSP é, então, uma componente chave na arquitetura de alta escala para aplicações baseadas na *Web*, provendo um modo

simplificado e dinâmico de gerar páginas *web* que contêm um código dinamicamente gerado. Ela utiliza *tags* equivalentes às do XML e *scriptlets* escritas na linguagem de programação Java, para encapsular a lógica que gera o conteúdo da página.

Todas estas *tags* (HTML ou XML) passam diretamente para a página de resposta. Por separar a lógica da página do seu design, a tecnologia JSP é, provavelmente, a forma mais fácil e mais rápida de construção de aplicações baseadas na *Web*. A tecnologia JSP é uma extensão da tecnologia *JavaServlet*. Esta estrutura pode ser utilizada para ampliar as capacidades de um servidor *Web*, com sobrecarga, manutenção e apoio mínimos (BONFIN JUNIOR, 2002) e (GIGAFLOPS, 2010).

### 2.1.8.2 - Compilação

Um ponto importante é como uma página JSP é compilada quando o usuário a carrega em *Web* browser. O processo de compilação é ilustrado na figura 14.

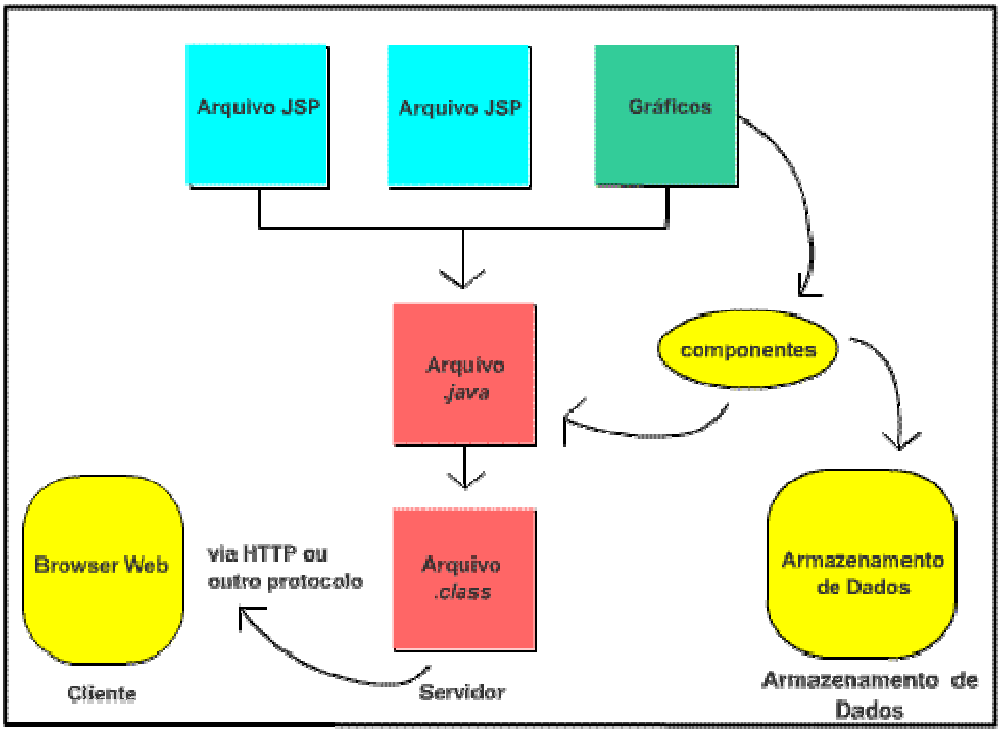


**Figura 14 - Processo de compilação. Fonte: (GIGAFLOPS, 2010).**

Uma aplicação JSP é, normalmente, uma coleção de arquivos JSP, arquivos HTML, gráficos e outras fontes. Quando o usuário carrega a página pela primeira vez, os arquivos que constituem a aplicação são todos traduzidos, em conjunto, sem qualquer dado dinâmico, para um único arquivo fonte Java (.java), com o nome definido pela sua aplicação JSP. Em seguida, o arquivo .java é compilado em um arquivo .class. Normalmente, o arquivo .java é uma *servlet* Java, a qual está de acordo com a API Java *Servlet*. Estes três passos são conhecidos como "tempo de tradução" (*translation time*).

Quando um usuário realiza uma solicitação (*request*) à aplicação JSP, colocando algo em um campo do formulário e selecionando um botão *Submit*, um ou mais de um dos componentes da aplicação (*bean*, *enterprise bean* ou *servlet*) manipula os dados que o usuário submeteu ou recupera os dados dinamicamente de um repositório de dados e retorna os dados para um arquivo .java, o qual é recompilado em um arquivo .class. O arquivo .class, sendo uma *servlet* Java, retorna os dados para o *Web* browser do cliente através do seu método *service*. Quando o usuário faz uma nova solicitação, o componente obtém ou manipula os dados novamente e os retorna para o arquivo .java, o qual é novamente compilado em um arquivo .class. Essa etapa é conhecida como "tempo de solicitação" (*request time*) (CORESERVLETS, 2010).

Quando uma página JSP é requisitada pelo cliente através de um Browser, esta página é executada pelo servidor, e a partir daí será gerada uma página HTML que será enviada de volta ao browser do cliente. A figura 15 ilustra o funcionamento do processo de compilação.

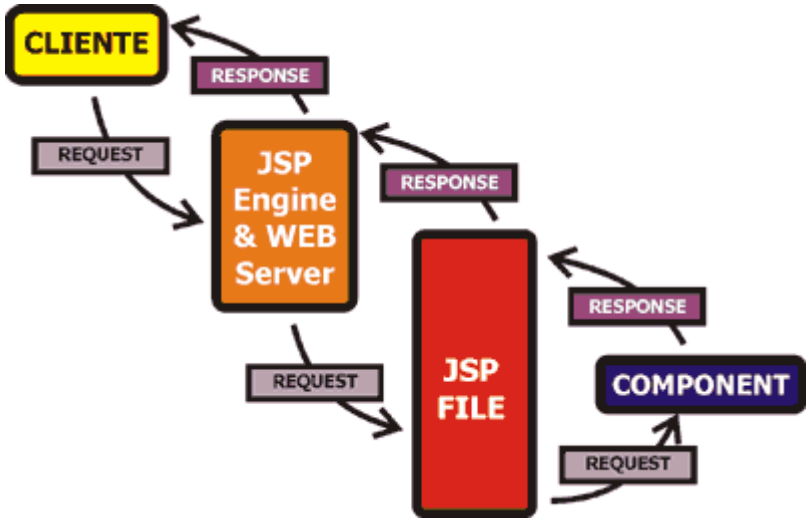


**Figura 15 - Processo de compilação JSP. Fonte: (CORESERVLETS, 2010).**

Quando o cliente faz a solicitação de um arquivo JSP, é enviado um *object request* para a JSP *engine*. A JSP *engine* envia a solicitação de qualquer componente (podendo ser um *JavaBeans component*, *servlet* ou *enterprise Bean*) especificado no arquivo. O componente controla a requisição possibilitando a recuperação de arquivos em banco de dados ou outro dado armazenado, em seguida, passa o objeto *response* de volta para a JSP *engine*. A JSP *engine* e o WEB server enviam a página JSP revisada de volta para o cliente, onde o usuário pode visualizar os resultados através do WEB browser. O protocolo de comunicação usado entre o cliente e o servidor pode ser HTTP ou outro protocolo.

### 2.1.9 - *Java Server Faces* (JSF)

JSF é uma tecnologia que incorpora características de um framework MVC para WEB e de um modelo de interfaces gráficas baseado em eventos. Por ser baseado no padrão de projeto MVC, uma de suas melhores vantagens é a clara separação entre a visualização e regras de negócio (modelo). A figura 16 mostra o framework baseado em aplicações *Web*.

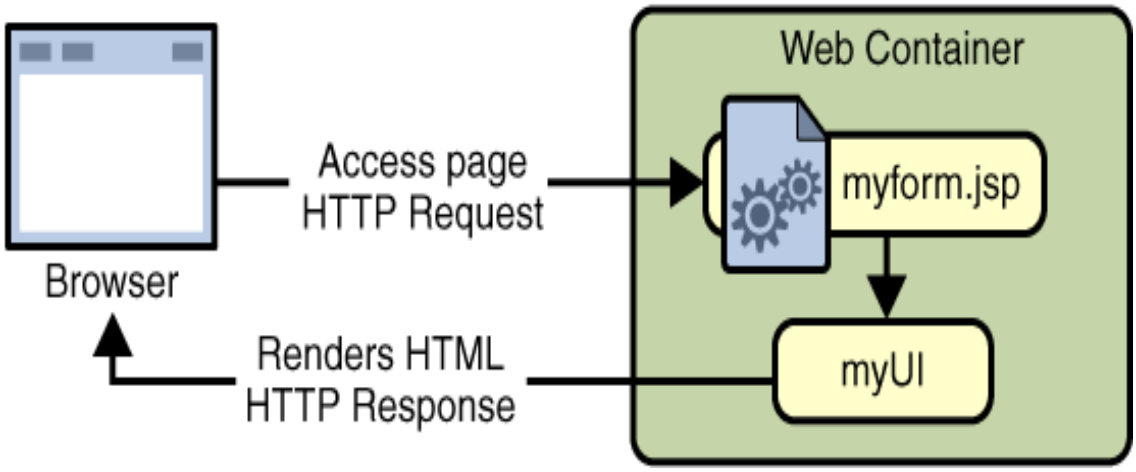


**Figura 16 - Framework do lado servidor baseado em aplicações WEB. Fonte: (GEARY; HORSMANN, 2004).**

A idéia do padrão MVC é dividir uma aplicação em três camadas: modelo, visualização e controle. O modelo é responsável por representar os objetos de negócio, manter o estado da aplicação e fornecer ao controlador o acesso aos dados. A visualização representa a interface com o usuário, sendo responsável por definir a forma como os dados serão apresentados e encaminhar as ações dos usuários para o controlador. Já a camada de controle é responsável por fazer a ligação entre o modelo e a visualização, além de interpretar as ações do usuário e as traduzir para uma operação sobre o modelo, onde são realizadas mudanças e, então, gerar uma visualização apropriada (SILVEIRA COSENTINO, 2009).

#### 2.1.9.1 - Padrão MVC segundo JSF

No JSF, o controle é composto por um *servlet* denominado *FacesServlet*, que é responsável por receber requisições da WEB, redirecioná-las para o modelo e então enviar uma resposta. Os arquivos de configuração são responsáveis por realizar associações e mapeamentos de ações e pela definição de regras de navegação. Os manipuladores de eventos são responsáveis por receber os dados vindos da camada de visualização, acessar o modelo, e então devolver o resultado para o *FacesServlet*.

O modelo representa os objetos de negócio e executa uma lógica de negócio ao receber os dados vindos da camada de visualização. Finalmente, a visualização é composta por *component trees* (hierarquia de componentes *User Interface* (UI)), tornando possível unir um componente ao outro para formar interfaces mais complexas. A figura 17 mostra a arquitetura JSF baseada no modelo MVC.

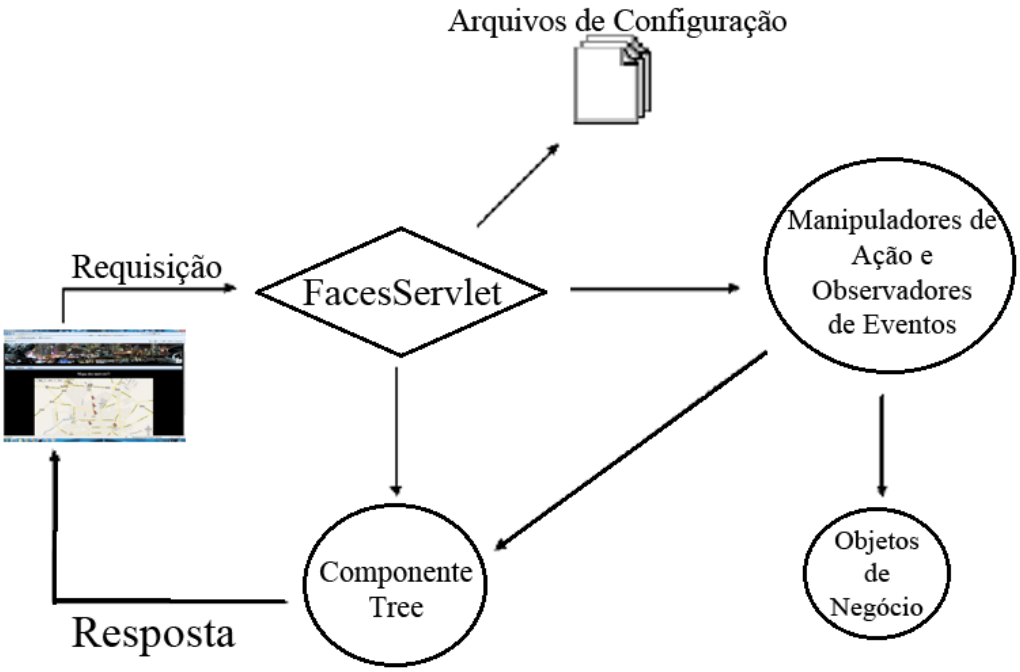


**Figura 17 - Arquitetura JSF baseada no modelo MVC. Fonte: (SILVEIRA COSENTINO, 2009).**

### 2.1.9.2 - Ciclo de Vida

O ciclo de vida do processo de requisição do JSF basicamente é composto por:

- **Fase de Restauração da Visão (*Restore View Phase*):** quando uma página JSF é requisitada é criada uma árvore de componentes que representam as tags JSF contidas na página além dos conversores, validadores e tratadores de eventos associados. Esta árvore de componentes é armazenada em um objeto

da classe *FacesContext* para ser usado na próxima requisição. Se a página está sendo requisitada pela primeira vez então a próxima fase é a última (montagem da página de resposta). Caso contrário passa-se à próxima fase;

- **Fase de Aplicação dos Valores da Requisição (*Apply Request Values Phase*): c**ada componente da árvore de componentes é atualizado com os valores contidos na requisição. A atualização inclui uma conversão de tipo. Se houver um problema de conversão (*type mismatch*) então será adicionada uma mensagem de erro para cada componente com problema de conversão e o próximo ciclo será o último (montagem da resposta);
- **Fase de Processamento das Validações (*Process Validations Phase*): n**esta fase os dados fornecidos pelo usuário são validados executando-se um método de um *bean* indicado pelo atributo "*validator*" da *tag* JSF. As validações normalmente são sintáticas;
- **Fase de Atualização dos Valores do Modelo (*Update Model Values Phase*):** nesta fase os valores dos componentes JSF são copiados para os atributos do *bean* associado à página usando-se os métodos set do *bean*. Somente após esta fase é que o *bean* conterá nos seus atributos os dados digitados pelo usuário. A única certeza sobre estes dados é que eles são sintaticamente corretos uma vez que já foram validados na fase anterior;
- **Fase da Invocação da Aplicação (*Invoke Application Phase*):** nesta fase o usuário submeteu um formulário selecionando um botão ou *link*. Em ambos os casos um método será executado (normalmente acessando algum objeto da camada lógica) e deverá ser retornada uma string indicando o resultado lógico da execução do método. Este valor será usado na fase seguinte para decidir qual página deverá ser montada e enviada de volta para o browser;
- **Fase da Renderização da Resposta (*Render Response Phase*):** a página de resposta ao usuário é montada.

A figura 18 ilustra com detalhe o ciclo de vida de um FSF.

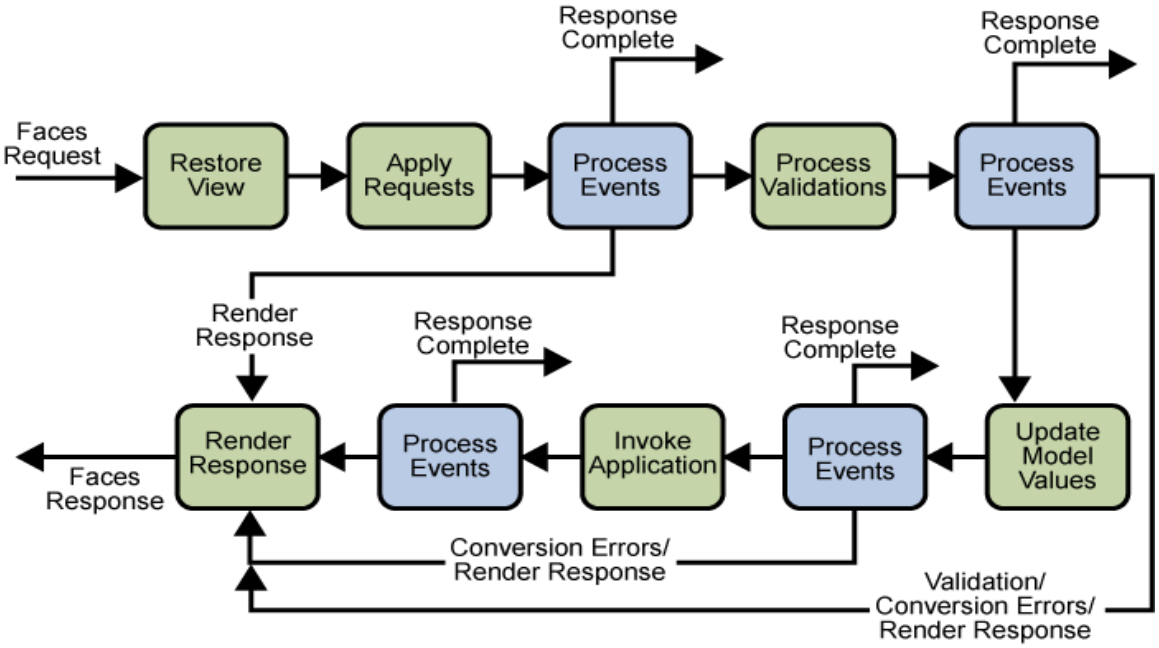


**Figura 18 - Ciclo do JSF. Fonte: (BERGSTEN, 2004).**

### 2.1.9.3 - Características e Vantagens

Java Server Faces oferece ganhos no desenvolvimento de aplicações WEB por diversos motivos que são:

- Permite que o desenvolvedor crie UIs (*User Interfaces*) através de um conjunto de componentes UIs prédefinidos;
- Fornece um conjunto de *tags* JSP para acessar os componentes;
- Reusa componentes da página;
- Associa os eventos do lado cliente com os manipuladores dos eventos do lado servidor (os componentes de entrada possuem um valor local representando o estado no lado servidor);
- Fornece separação de funções que envolvem a construção de aplicações WEB.

Embora *JavaServer Faces* forneça *tags* JSP para representar os componentes em uma página, ele foi projetado para ser flexível, sem limitar-se a nenhuma linguagem *markup* em particular, nem a protocolos ou tipo de clientes. Ele também permite a criação de componentes próprios a partir de classes de componentes. JSF possui dois principais componentes: Java APIs para a representação de componentes UI e o gerenciamento de seus estados, manipulação/observação de eventos, validação de entrada, conversão de dados, internacionalização e acessibilidade; e *taglibs* JSP que expressam a interface JSF em uma página JSP e que realizam a conexão dos objetos no lado servidor.

## 2.2 – XML (*Extensible Markup Language*)

O XML é uma tecnologia que tem ao seu redor outras tecnologias que a complementam e a faz muito maior e com possibilidades muito mais amplas. O XML é importante no mundo globalizado atual, pois pode compartilhar as informações de uma maneira segura, fácil e confiável.

Ao redor do XML está um mundo de tecnologias, facilidade no trabalho com os dados e um avanço na hora de tratar a informação. Enfim o XML não é só uma linguagem e uma sintaxe, é sim várias linguagens com várias sintaxes (FURTADO JUNIOR, 2010), (W3SCHOOLS, 2010) e (ALVAREZ, 2010).

### 2.2.1 – História do XML

O XML provém de uma linguagem que a IBM inventou por volta dos anos 70. A linguagem de IBM chama-se *General Markup Language* (GML) e surgiu da necessidade que tinham na empresa de armazenar grandes quantidades de informação sobre temas diversos.

### 2.2.2 – Objetivos e vantagens do XML

Os principais objetivos da linguagem XML são:

- Ser idêntico ao HTML na hora de servir, receber e processar a informação;
- Ser formal e conciso do ponto de vista dos dados a maneira de salvá-los;
- Que fosse extensível;
- Fácil de ler e editar;
- Fácil de implantar, programar e aplicar aos distintos sistemas;

As principais vantagens do XML são:

- **Comunicação de dados**: se a informação se transfere em XML, qualquer aplicação poderia escrever um documento de texto com os dados que estava manejando em formato XML e outra aplicação receber a informação e trabalhar com ela;
- **Migração de dados**: para mover os dados de uma base de dados à outra seria muito simples se as duas trabalhassem em formato XML.

### 2.2.3 - *Definition Type Document* (DTD) ou XML *Schema*

Um documento XML pode conter muitos tipos de informação, ou seja, pode haver muitas linguagens escritas em XML e isso depende muito do tipo de usuários.

Podem-se criar infinitas linguagens a partir do XML, para especificar cada um de seus usos. São umas linguagens que servem para definir outras linguagens, ou seja, são metalinguagens. Estas são definidas especificando quais etiquetas podem ou devem encontrar nos documentos HTML, em que ordem, dentro de quais outras e

especificar os atributos que podem ou devem ter cada umas dessas etiquetas. Existem duas metalinguagens com as quais podem ser definidas as linguagens que podem ser obtidas a partir de XML, o DTD e o XML *Schema* (XMLDESIGNER, 2010).

O DTD (*Definition Type Document*), tem uma sintaxe especial, diferente da de XML, que é simples. Para evitar o DTD, que tem uma sintaxe muito especial, tentou-se encontrar uma maneira de escrever em XML a definição de outra linguagem XML. Definiu-se então a linguagem XML *Schema* e funciona bem, embora possa chegar a ser um pouco mais complicado que especificá-la em DTD.

Um detalhe importante de assinalar na hora de falar dos DTD ou XML *Schema* é que estas linguagens também permitem comprovar a integridade dos dados em qualquer momento. As metalinguagens de XML servem para pegar um documento em formato XML e comprovar que os dados que ele inclui são válidos, comprovando se o que temos no XML concorda com o que teríamos que ter. Isso pode ser feito ao ler o documento, se não forem válidos tira-se uma mensagem de erro e se detém o processo do documento. Se forem válidos, fazemos o que for sem ter que nos preocuparmos pela integridade dos dados (FURTADO JUNIOR, 2010), (W3SCHOOLS, 2010) e (XMLDESIGNER, 2010).

## 2.3 - *Web Service* (WS)

A *Web* teve início estimulando as interações humanas com texto e gráficos. As pessoas usam a *Internet* diariamente para acompanhar suas ações, comprar produtos e ler as últimas notícias. O nível de interação disponível é satisfatório para muitos destes propósitos. Mas esta *Web* não suporta muito bem a interação entre softwares, principalmente a transferência de grandes quantidades de dados. Um método mais eficiente é fundamental para permitir que os aplicativos interajam diretamente entre eles, executando automaticamente instruções que, da forma tradicional, deveriam ser fornecidas manualmente por um ser humano através de um browser.

As pessoas e as empresas que fazem negócios pela rede necessitam de uma forma de publicar links para seus aplicativos e seus dados, mais ou menos da mesma forma que eles publicam seus links para as páginas *Web*. Os aplicativos baseados na *Internet* precisam ser capazes de encontrar outros aplicativos baseados na *Internet*, acessá-los e interagir com eles automaticamente. Os WS sofisticam o uso da *Internet* permitindo esta comunicação programa-a-programa. Com a difusão dos WS, aplicativos sediados em distintos locais da *Internet* e provenientes de distintos fornecedores podem ser diretamente integrados e interconectados, como se fizessem parte de um único e grande sistema de informação. A figura 19 mostra a interação de um sistema com *web services*.

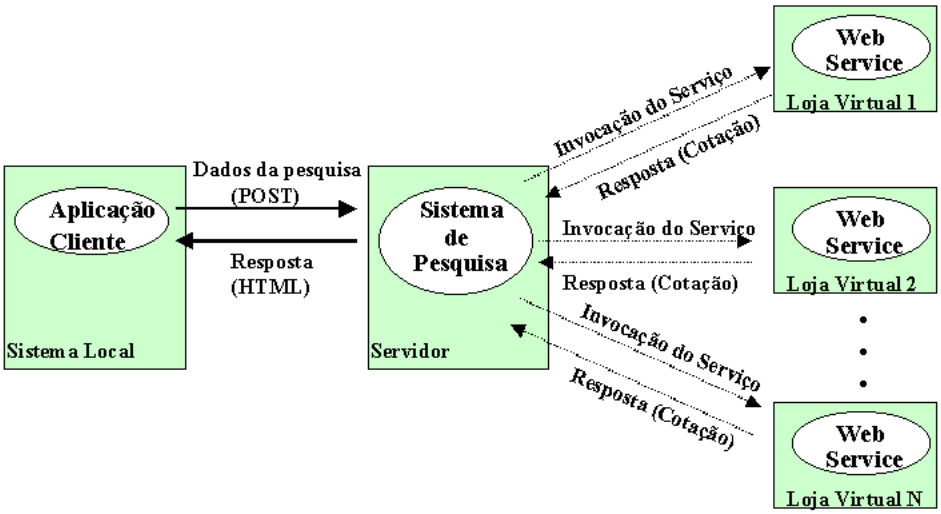


**Figura 19 - Interação de sistema com Web Services. Fonte: (CUNHA, 2010).**

Os WS são aplicações XML mapeadas sobre programas, objetos, bases de dados e até sobre complexas regras de negócio. Usando um documento XML criado na forma de uma mensagem, um programa envia uma requisição a um *Web* Service pela rede e, opcionalmente, recebe uma resposta, também na forma de um documento XML. Os padrões relacionados aos WS definem o formato da mensagem, especificam a interface para onde a mensagem é enviada, descrevem as convenções para o mapeamento do conteúdo de uma mensagem na entrada e

na saída de programas que implementam o serviço e definem mecanismos para publicar e encontrar interfaces para WS (ALONSO et al., 2004). Esta tecnologia pode ser usada de muitas formas. Os WS podem ser executados em computadores de mesa e de mão para acessar aplicativos via *Internet*, como sistemas de reserva e de acompanhamento de pedidos. Os WS também podem ser usados para a integração de empresas de B2B *(Business-to-Business*), conectando aplicativos de diversas organizações de uma mesma cadeia de produção. Os WS também podem resolver problemas mais amplos de integração de aplicativos empresariais (EAI - *Enterprise Application Integration*), conectando diversos aplicativos de uma única organização com diversos outros aplicativos, tanto dentro quanto fora de um *firewall*. Em todos estes casos, as tecnologias dos WS funcionam como uma cola padrão para unir todas as peças de software.

A Figura 20 ilustra como os WS apresentam uma forma padronizada de interfaceamento entre sistemas de retaguarda (*back-end*), como sistemas de gerenciamento de bancos de dados, .NET, J2EE e CORBA, objetos, adaptadores de pacotes ERP *(Enterprise Resource Planning* - Planejamento de Recursos Empresariais), *brokers* (intermediários) de integração e outros. As interfaces WS recebem uma mensagem padronizada em formato XML de um ambiente de rede, transforma os dados XML em um formato compreendido por um sistema especifico e, opcionalmente, devolve uma mensagem de resposta. Podem-se criar implementações de WS usando qualquer linguagem de programação, sistema operacional ou *middleware*.

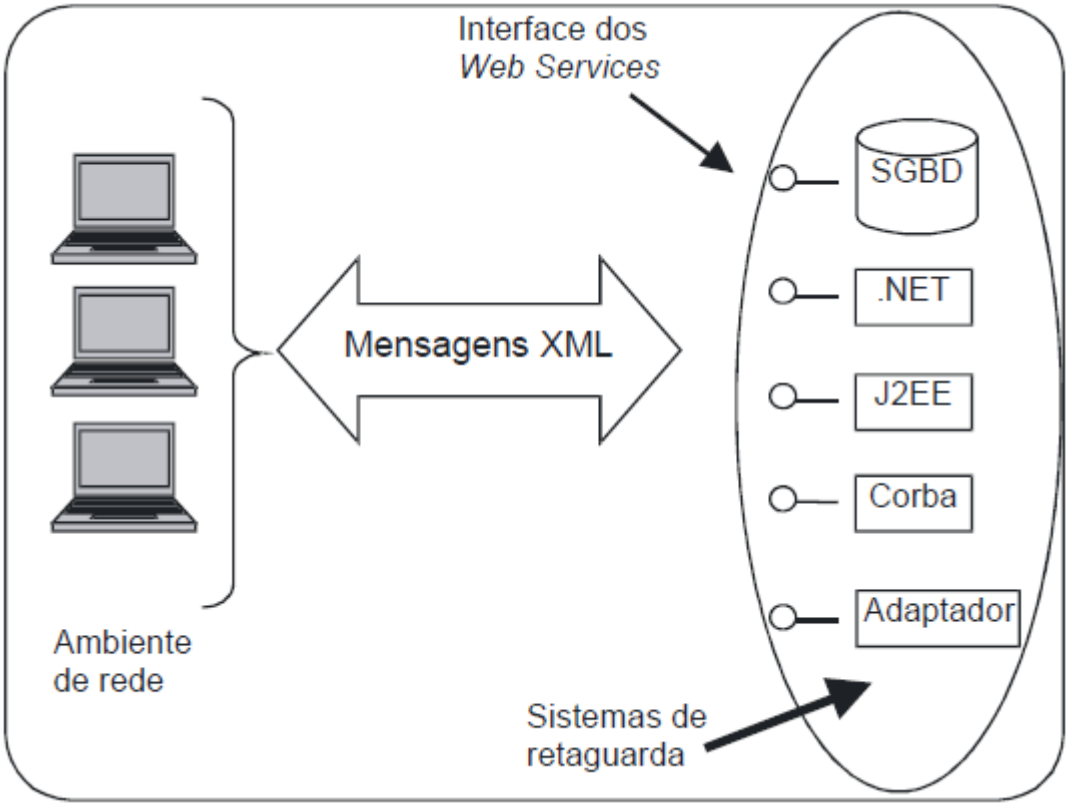


**Figura 20 - Interface do WS com sistemas de retaguarda. Fonte: (LUCCA, 2003).**

Os WS combinam as características de execução de programas com características de abstração que possui a *Internet*. As tecnologias atuais usadas na *Internet* obtêm sucesso em parte porque estão definidas com um nível tal de abstração que permitem a compatibilidade entre quaisquer sistemas operacionais, hardware e software. A infra-estrutura da *Internet* baseada em WS explora este nível de abstração e associa significado aos dados que trafegam por ela. Ou seja, os WS definem não somente os dados, mas também como processá-los e mapeá-los na entrada e na saída dos aplicativos subjacentes.

### 2.3.1 - Arquitetura de *Web Service*

Uma *Web service* nada mais é que um serviço disponibilizado na *Internet*, descrito via WSDL, registrado via UDDI, acessado com o auxílio de SOAP (*Simple Object*

*Access Protocol*) e com os dados transmitidos sendo representados em XML (CUNHA, 2010). SOAP é um protocolo projetado para invocar aplicações remotas através de RPC (*Remote Procedure Calls* - Chamadas Remotas de Procedimento) ou trocas de mensagens, em um ambiente independente de plataforma e linguagem de programação. SOAP é, portanto, um padrão normalmente aceito para utilizar-se com *Web Services*. Desta forma, pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização de uma linguagem (XML) e mecanismo de transporte (HTTP) padrões (CUNHA, 2010). A figura 21 ilustra os elementos de uma mensagem SOAP. Estes elementos são:

- **Envelope**: toda mensagem SOAP deve contê-lo. É o elemento raiz do documento XML. O envelope pode conter declarações de *namespaces* e também atributos adicionais como o que define o estilo de codificação (*encoding style*). Um *"encoding style*" define como os dados são representados no documento XML.
- **Header**: é um cabeçalho opcional. Ele carrega informações adicionais, por exemplo, se a mensagem deve ser processada por um determinado nó intermediário (é importante lembrar que, ao trafegar pela rede, a mensagem normalmente passa por diversos pontos intermediários, até alcançar o destino final). Quando utilizado, o *Header* deve ser o primeiro elemento do envelope.
- **Body**: este elemento é obrigatório e contém o *payload*, ou a informação a ser transportada para o seu destino final. O elemento *Body* pode conter um elemento opcional *Fault*, usado para carregar mensagens de status e erros retornadas pelos "nós" ao processarem a mensagem.

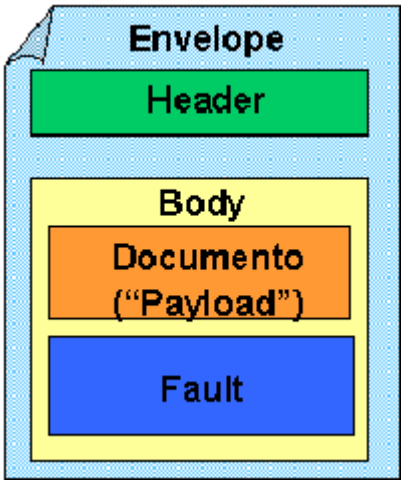


**Figura 21 - Estrutura de uma mensagem SOAP. Fonte: (CUNHA, 2010).**

WSDL (*Web Services Description Language*) é a linguagem de descrição de *web* services baseada em XML. Ela permite, através da definição de um vocabulário em XML, a possibilidade de descrever serviços e a troca de mensagens. Mais especificamente é responsável por prover as informações necessárias para a invocação da *web service*, como sua localização, operações disponíveis e suas assinaturas (CUNHA, 2010).

UDDI (*Universal Description, Discovery and Integration*) é uma das tecnologias que possibilitam o uso de *web* services. Uma implementação de UDDI corresponde a um *Web Service Registry*, que provê um mecanismo para busca e publicação *web* services. Um UDDI *registry* contém informações categorizadas sobre os serviços e as funcionalidades que eles oferecem, e permite a associação desses serviços com suas informações técnicas, geralmente definidas usando WSDL. Como dito anteriormente, o arquivo de descrição em WSDL descreve as funcionalidades do *web service*, a forma de comunicação e sua localização. Devido ao modo de acesso, um UDDI *registry* também pode ser entendido como uma *web* service. A especificação UDDI define uma API baseada em mensagens SOAP, com uma descrição em WSDL da própria *web service* do servidor de registro. A maioria dos servidores de registro UDDI também provê uma interface de navegação por *browser* (CUNHA, 2010).

A arquitetura de *web services* se baseia na interação de três entidades: provedor do serviço (*service provider*), cliente do serviço (*service requestor*) e servidor de registro (*service registry*). De uma forma geral, as interações são para publicação, busca e execução de operações. A figura 22 ilustra estas operações, os componentes envolvidos e suas interações.

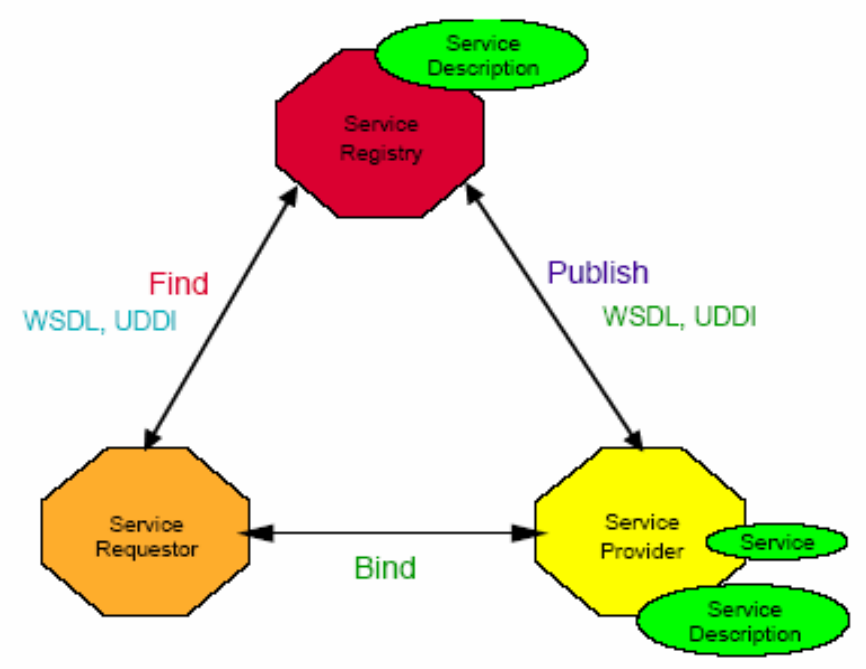


**Figura 22 – Arquitetura de *web services*. Fonte: (CUNHA, 2010).**

O provedor do serviço (*service provider*) representa a plataforma que hospeda a *web* service permitindo que os clientes acessem o serviço. O cliente do serviço (*service requestor*) é a aplicação que está procurando, invocando ou iniciando uma interação com o *web service*. O cliente do serviço pode ser uma pessoa acessando através de um browser ou uma aplicação realizando uma invocação aos métodos descritos na interface da *web service*. O *service registry* representa os servidores de registro e busca de *web* services baseados em arquivos de descrição de serviços que foram publicados pelos *service providers*. Os clientes (*service requestor*) buscam por serviços nos servidores de registro e recuperam informações referentes a interface de comunicação para os *web services* durante a fase de desenvolvimento ou durante a execução do cliente, denominados *static binding* e *dinamic binding*, respectivamente (CUNHA, 2010).

## 2.4 - *Busines to Consumer* (B2C)

O B2C (*Business to Consumer*) é o segmento dentro do comércio eletrônico que abrange qualquer transação em que uma companhia ou organização vende seus produtos ou serviços para as pessoas que navegam pela *Internet*. Esse segmento se assemelha muito com as lojas que fazem venda direta ao consumidor (varejo) através de catálogos (AMSDESIGN, 2010).

O segmento B2C se apresenta tipicamente na *Web* na forma de lojas virtuais onde o consumidor pode navegar no site da loja e adquirir bens ou produtos ofertados pelo site. Alguns exemplos de lojas virtuais do segmento B2C são:

- **Livrarias virtuais**: Amazon (www.amazon.com);
- **Venda de passagens aéreas**: TAM (www.tam.com.br);
- **Venda de serviços bancários**: Banco do Brasil (www.bb.com.br);
  Bradesco (www.bradesco.com.br);
  Itaú (www.itau.com.br).

O volume das transações de B2C tem crescido muito ultimamente e isto justifica o uso dessa tecnologia para o desenvolvimento do aplicativo deste projeto.

## 2.5 - *Google Maps* API

O *Google Maps* é um serviço gratuito fornecido pela empresa *Google*, usado para realizar pesquisas, visualização de mapas e imagens de satélite da Terra. O *Google Maps* possibilita a produção de um mapa personalizado completo, sendo possível marcar locais, adicionar vídeos, fotos e compartilhar todo este conteúdo na rede mundial de computadores (GOOGLE MAPS, 2010) e (LIU, 2010). Quando o *Google*

*Maps* foi lançado, em fevereiro de 2005, ainda em sua versão beta, tornou-se rapidamente uma referência em serviços de mapas na *Internet*. Com uma interface rica e intuitiva, a aplicação permitia acesso a uma enorme base de dados contendo inúmeras imagens de satélite, mapas de cidades, bairros, ruas e avenidas dos Estados Unidos. Com o tempo, novas localidades foram sendo adicionadas no sistema até que em meados de maio de 2007, a *Google* finalmente disponibilizou consultas de endereços no Brasil, sendo que, em outubro de 2007, uma versão estendida para o público brasileiro foi oferecida, com a possibilidade de se localizar restaurantes, hotéis, traçar rotas, dentre outras utilidades (AZEVEDO, 2010). A interface inicial do *Google Maps* é mostrada na figura 23.

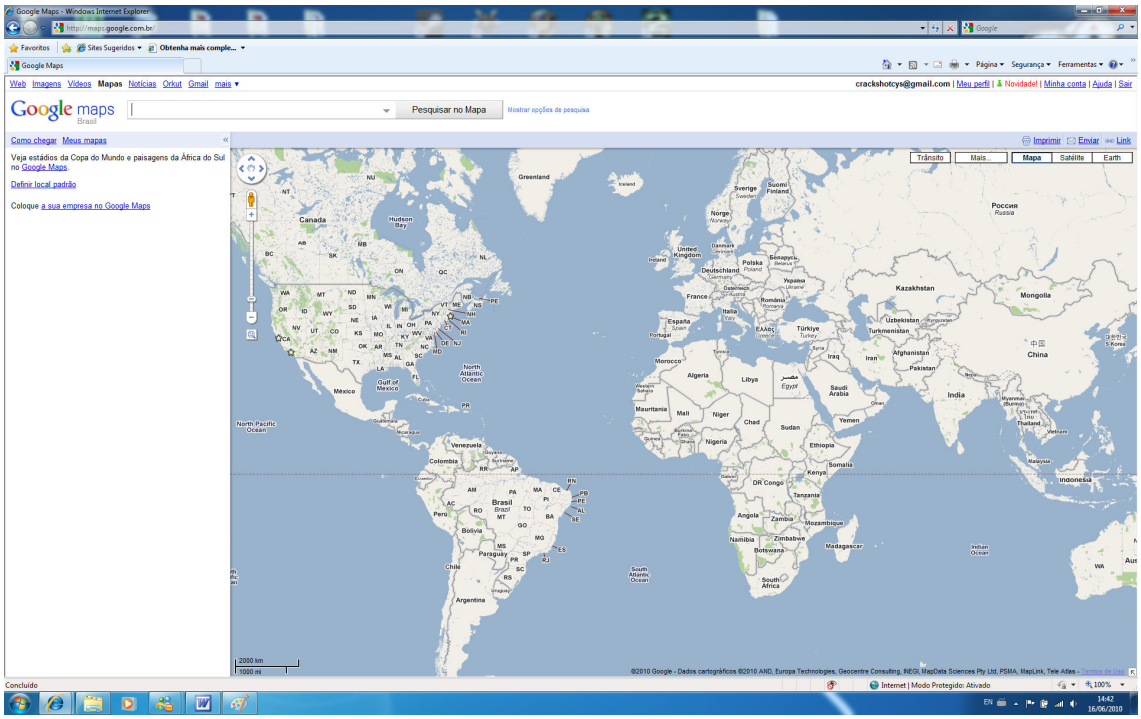


**Figura 23 - Interface inicial do *Google Maps*. Fonte: (GOOGLE MAPS, 2010).**

A API do *Google Maps* é uma interface de desenvolvimento para aplicações baseadas no próprio *Google Maps*, permitindo criar aplicativos inovadores de mapeamento para a plataforma da *Web*. Essa API ajuda a integrar mapas e geo-codificação aos sites, possibilitando que as aplicações com conteúdo geo-

referenciado sejam facilmente apresentadas em qualquer navegador (GOOGLE DEVELOPER GUIDE, 2010).

Um passo importante na disponibilização em escala global de informações geográficas gratuitas foi dado pela Google quando lançou o site *Google Maps*. Isto leva a afirmar que se trata de um marco histórico para a disponibilização de informações geográficas gratuitas não é o site em si, mas sim o fato de, paralelamente, a Google ter criado uma forma dos utilizadores das informações geográficas poderem incluir os mapas disponibilizados pela Google em seus próprios sites, sem a necessidade de nenhum componente extra no servidor *Web* como as APIs (FARIA, 2006). A primeira versão da API do *Google Maps* foi disponibilizada em junho de 2005. A API consiste basicamente de um pequeno conjunto de classes *JavaScript* que fornecem uma interface necessária para que o usuário possa construir aplicações para exibir os mapas, realizar consultas por endereços, realizar funções de zoom, acrescentar pontos de referência ou descrições no mapa, dentre outras possibilidades. A figura 24 mostra a localização da FEMA (Fundação Educacional do Município de Assis) no *Google Maps*.


**Figura 24 – Localização da FEMA. Fonte: (GOOGLE MAPS, 2010).**

Para utilizar a API, é preciso solicitar ao Google uma chave única informando o endereço do site onde se deseja integrar os recursos do *Google Maps* no link: http://code.google.com/intl/pt-BR/apis/maps/signup.html. Alguns dos recursos disponíveis são: zoom; pan; visualizar imagens de satélites de diversos locais do planeta; incluir marcadores de locais; realizar overlay de arquivos KML; pesquisar por locais; traçar rotas; buscar endereços; inserir linhas; polígonos; etc. É importante frisar que todo este procedimento se dá dentro da "filosofia" AJAX, o que significa dizer que o usuário não tem que se preocupar com os detalhes (SOUZA NETO, 2009).

O AJAX surgiu em fevereiro de 2005, como sendo um conceito de navegação e atualização de páginas *Web*. Antes do surgimento do AJAX, exigia-se que para cada solicitação em uma página *Web*, a página inteira fosse atualizada no navegador, ou seja, independentemente da atualização ser apenas de uma pequena parte da página, toda a página era recebido pelo servidor, redesenhada e retornada para o navegador, o que gerava mais tráfego de informações do que era necessário e conseqüentemente, tornando o processo mais lento. Sendo assim, com a utilização do AJAX pode-se trafegar apenas os dados que realmente foram atualizados em uma página *Web*, ganhando-se em qualidade e rapidez de processamento. AJAX também pode ser utilizado em conjunto com qualquer tecnologia Java para *Web* (LIMEIRA, 2006).

## 2.6 - *Apache Tomcat*

O *Tomcat*, desenvolvido pela Fundação *Apache*, é um servidor de aplicações *web*. Sua principal característica técnica é estar centrada na linguagem de programação Java, mais especificamente nas tecnologias de *Servlets* e de *Java Server Pages* (JSP). Ele está escrito em Java e, por isso, necessita que a versão *Java 2 Standard Edition* (J2SE) esteja instalada no mesmo computador onde ele será executado. No entanto, não basta ter a versão *runtime* de Java instalada, pois o *Tomcat* necessita compilar (e não apenas executar) programas escritos em Java. O servidor *Tomcat* tem a habilidade de converter automaticamente qualquer página JSP em um *servlet*

equivalente. Em outras palavras, o *Tomcat* é capaz de criar código fonte Java a partir de um documento HTML (APACHE TOMCAT, 2010).

A Fundação Apache, mais conhecida pelo seu servidor *web* de mesmo nome, permite como no caso do servidor Apache, que o *Tomcat* seja usado livremente, seja para fins comerciais ou não. Do ponto de vista técnico, *Tomcat* é a implementação referência das especificações das tecnologias de *servlets* e JSP criadas pela Sun. Do ponto de vista operacional, a principal finalidade das tecnologias de *servlets* e JSP é permitir a criação dinâmica de conteúdos. A dinâmica, em um cenário típico, funciona do seguinte modo:

- Um usuário, no seu browser, solicita algum documento (indicado por um URL) a um servidor *Tomcat*;
- O servidor, ao receber uma solicitação (URL) do usuário, executa o *servlet* ou JSP correspondente àquele URL (a associação entre URL e *servlet* ou JSP é especificada no arquivo web.xml). O conteúdo gerado pelo *servlet* ou JSP, normalmente um documento no formato HTML, é uma combinação de *tags* HTML (incluídos explicitamente) e o resultado de algum processamento (por exemplo, algoritmo Java e/ou acesso a um banco de dados);
- O usuário recebe o conteúdo gerado pelo servidor *Tomcat* e o exibe através do seu browser.

## 2.7 - Hibernate

O Hibernate é um *framework* incrível de mapeamento objeto/relacional para Java que facilita o desenvolvimento de aplicações que acessam bancos de dados, fazendo com que o programador se preocupe mais com o seu modelo de objeto e seus comportamentos, do que com as tabelas do banco de dados (BAUER; KING, 2004).

Além de mecanismo de mapeamento objeto/relacional, o Hibernate também pode trabalhar com um sistema de *cache* das informações do banco de dados, aumentando ainda mais a *performance* das aplicações. O esquema de cache do Hibernate é complexo e totalmente extensível, existindo diversas implementações possíveis, cada uma com suas próprias características. E junto com isso, ele também tem um gerenciador de versões próprio. Usando o Hibernate, o desenvolvedor evita o trabalho de escrever dúzias de código repetido para fazer as mesmas coisas, como "*inserts*", "*selects*", "*updates*" e "*deletes*" no banco de dados, além de ter opções para se montar buscas complexas em grafos de objetos e ainda uma saída para o caso de nada funcionar, usar SQL (*Structured Query Language*).

Com toda essa facilidade oferecida pelo Hibernate, seria praticamente impossível um desenvolvedor não utilizá-lo em seu projeto.

## 2.8 - Banco de dados – HSQLDB

O *Hypersonic SQL Database* (HSQLDB) é um projeto de banco de dados livre, escrito em Java, que permite a manipulação de banco de dados em uma arquitetura cliente-servidor, ou *standalone*. Uma grande vantagem de utilização do HSQLDB é a possibilidade de agregar o banco de dados ao pacote de nossas aplicações. O banco é multiplataforma e ocupa um pequeno espaço em disco. Outra característica do banco é a possibilidade de manipular o banco de dados em disco, memória ou em formato texto. Essa tecnologia é flexível e muito útil na construção de aplicações que manipulam banco de dados.

No núcleo do pacote estão o RDBMS (*Relational Database Management System* ) e o *driver* JDBC (*Java Database Connectivity*) que disponibilizam as principais funcionalidades do banco, que são: o gerenciador de banco de dados relacional e o *driver* para conexão através de aplicações Java. Além disso, o pacote contém um conjunto de componentes e ferramentas para execução do SGBD (Sistema de Gerenciamento de Bando de Dados). Através das ferramentas podemos criar estruturas de um banco de dados, acessar bancos de dados através de ferramentas para consulta, exportar e importar esquemas entre bancos de dados distintos. Além

de outras facilidades disponibilizadas para o desenvolvedor (SEVERO, 2008). Será feita uma descrição de cada um desses componentes:

- ***HSQLDB JDBC Driver***: o pacote de distribuição disponibiliza um *driver* padrão JDBC para conexão de aplicações Java com o SGBD. A conexão com o banco de dados segue um modelo de protocolo proprietário, mas que também pode realizar uma conexão via rede, através de protocolos *Internet*;
- ***Database Manager***: são disponibilizadas duas versões de ferramentas para gerenciamento de banco de dados. Uma ferramenta escrita usando AWT (*Abstract Window Toolkit*) e outra versão usando *Swing*. Eles são ferramentas gráficas para visualização do esquema do banco de dados, conjunto de tabelas e submissão de instruções SQL. A versão AWT pode ser executada como um *Applet* dentro de um navegador;
- ***Transfer Tool***: é uma ferramenta utilizada para transferências de esquemas SQL ou dados de uma fonte JDBC para outra. Essa ferramenta é bastante útil para realizar uma migração de banco de dados, transferindo esquemas e o conjunto de dados entre duas tecnologias distintas;
- ***Query Tool***: a finalidade dessa ferramenta é prover ao desenvolvedor um software para interação com o SGBD através do envio de instruções SQL a partir de uma linha de comando, ou através de um arquivo texto contendo um conjunto de instruções. A ferramenta apresenta um *shell* interativo ao usuário;
- ***SQL Tool***: outra ferramenta do pacote para construção e submissão de instruções SQL ao banco de dados.

O HSQLDB é uma ferramenta muito flexível para manipulação de banco de dados e uma excelente ferramenta para quem trabalha com Java. Sendo assim, este foi o banco de dados utilizado nesse trabalho.

# CAPÍTULO 3

# DESENVOLVIMENTO DO APLICATIVO

Neste capítulo será apresentada a modelagem do problema, onde foram divididos por módulos para facilitar o entendimento das fases a serem desenvolvidas, e a sua especificação. Para a realização deste trabalho foram executados procedimentos de especificação e implementação visando o cumprimento dos objetivos.

## 3.1 - Descrição do Problema

Foi desenvolvido neste trabalho um aplicativo *web* utilizando as tecnologias Java para o setor imobiliário. Este aplicativo tem a funcionalidade de que os usuários possam se cadastrar e cadastrar seus imóveis. Outra funcionalidade do aplicativo é que os imóveis cadastrados podem ser visualizados e localizados em um mapa através da *API Google Maps*. O aplicativo é um sistema *web* com um servidor de serviços *web*, desenvolvido na linguagem JSF (*Java Server Faces*) com comunicação XML com o Servidor da Google através da *Google Maps* API para manipulação de dados espaciais e com o banco HSQLDB.

A figura 25 mostra a visão geral de um sistema *web*.

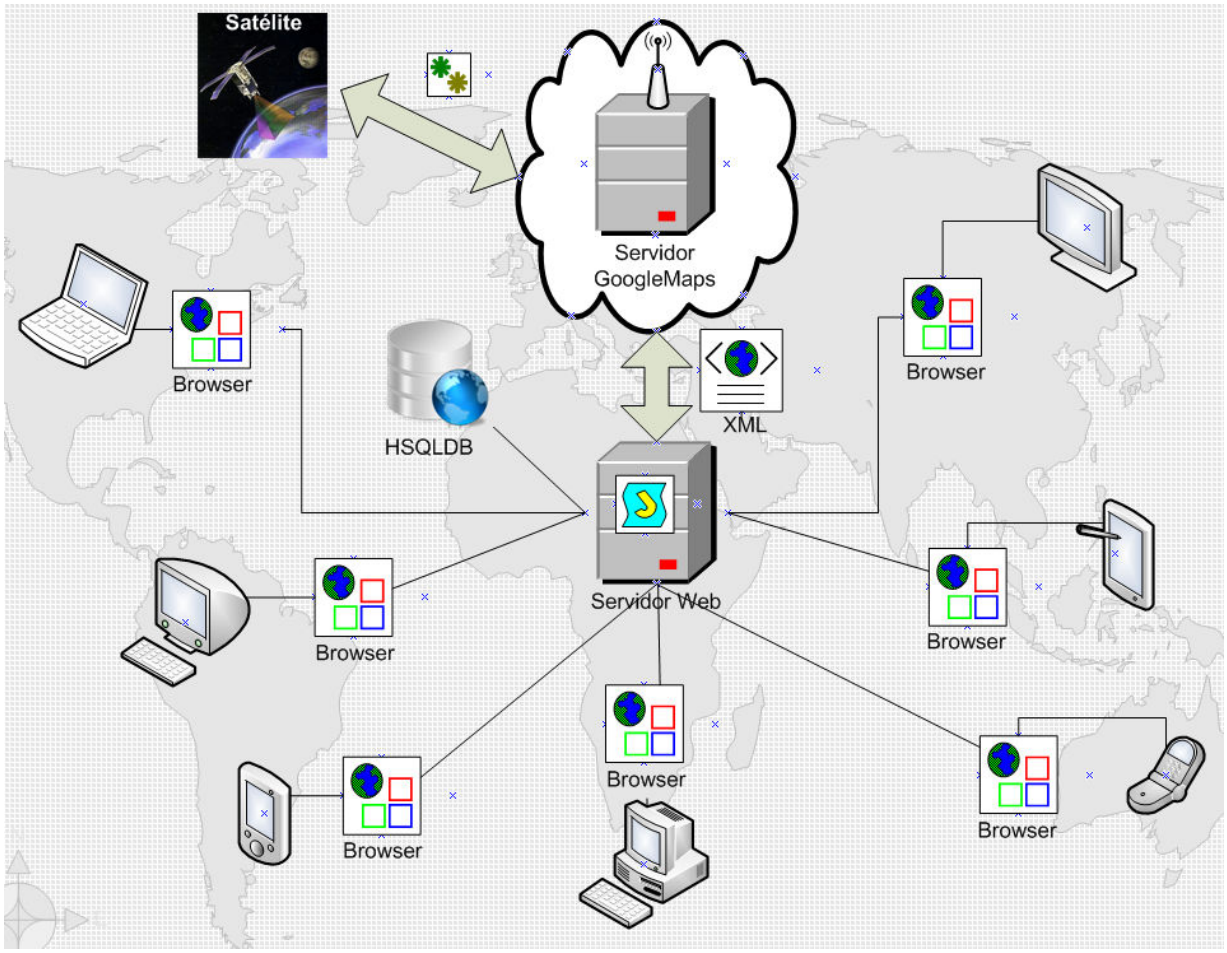


**Figura 25 - Visão Geral do Sistema *Web*.**

## 3.2 - Modelagem do Problema

A modelagem é de suma importância, pois ela serve para dar uma idéia geral do problema que será abordado. A figura 26 mostra a modelagem do problema, e ela foi dividida em quatro módulos para facilitar o desenvolvimento de cada parte que constitui o aplicativo.

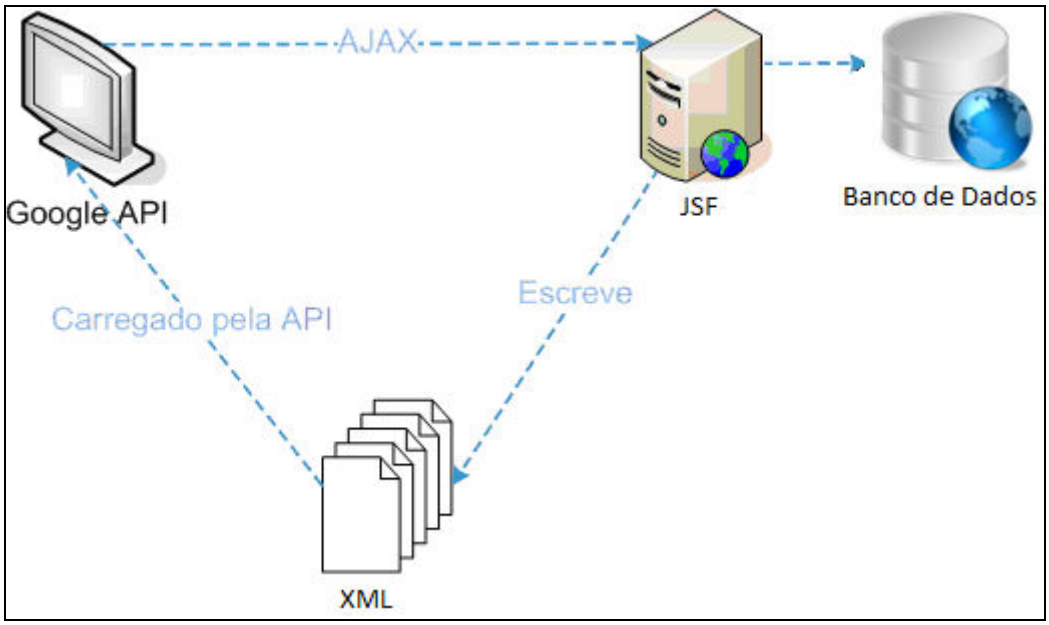


**Figura 26 – Modelagem do problema.**

**Módulo 1 - Criação do Banco de dados.**

Neste módulo foi criado o banco de dados HsqlDB, pois apresenta melhor resposta em referência a outros gratuitos para este tipo de problema e também tem oferecido um bom suporte a linguagem Java e ao Hibernate.

**Módulo 2 - Criação do Sistema e Interface do Usuário**

Neste módulo foi desenvolvido o sistema *web* em JSF, utilizando como ferramenta o *NetBeans*, que irá receber os dados enviados através do HTTP, e fazer o processamento das informações e efetuar o cadastro dos dados no banco de dados. Essas informações são atribuídas a qualquer usuário que possuir acesso a *internet*. E também será desenvolvido uma interface de uso simples e eficaz utilizando AJAX com *JavaScript*.

**Módulo 3 - Implementação do *Google Maps* API.**

Para a utilização de mapas no sistema como forma de inserção e pesquisa de imóveis por parte dos usuários lançou-se mão de um serviço fornecido pelo Google no qual é possível a visualização de mapas de todas as regiões do mundo. Para a utilização desse serviço é necessário o uso de uma API fornecida pela empresa GOOGLE. Portanto, neste módulo foi realizado a implementação de uma interface entre a API do *Google Maps* e o sistema.

**Módulo 4 - Comunicação XML.**

Neste módulo foi desenvolvida toda a comunicação entre Sistema, *Google Maps* e Banco de dados.

### 3.3 – Especificação

Para fazer a especificação deste aplicativo foi utilizada uma metodologia orientada a objetos, representada em diagramas UML (*Unified Modeling Language*), utilizando como ferramenta o *ArgoUML*. O primeiro diagrama utilizado na especificação é o de casos de uso, seguido pelo diagrama de classes, diagrama de atividades e por último os diagramas de seqüência especificando o funcionamento do aplicativo.

#### 3.3.1 – Diagrama de casos de uso

O aplicativo possui seis casos de uso, como mostra a figura 27.

- **Manter Clientes:** responsável por todas as operações relacionadas ao cliente.

- **Manter Imóveis:** responsável por todas as operações relacionadas ao imóvel.
- **Incluir:** responsável por armazenar objetos novos no banco de dados.
- **Pesquisar:** responsável por recuperar os objetos armazenados no banco de dados.
- **Excluir:** responsável por excluir o objeto anteriormente pesquisado.
- **Atualizar:** responsável por atualizar o objeto anteriormente pesquisado.
- **Visualizar Mapa:** responsável pela visualização do Mapa com a localização de todos os imóveis cadastrados no aplicativo através da API do *Google Maps*.
- **Acessar WebService:** responsável pela comunicação entre o aplicativo e diversos sistemas via *web services*.

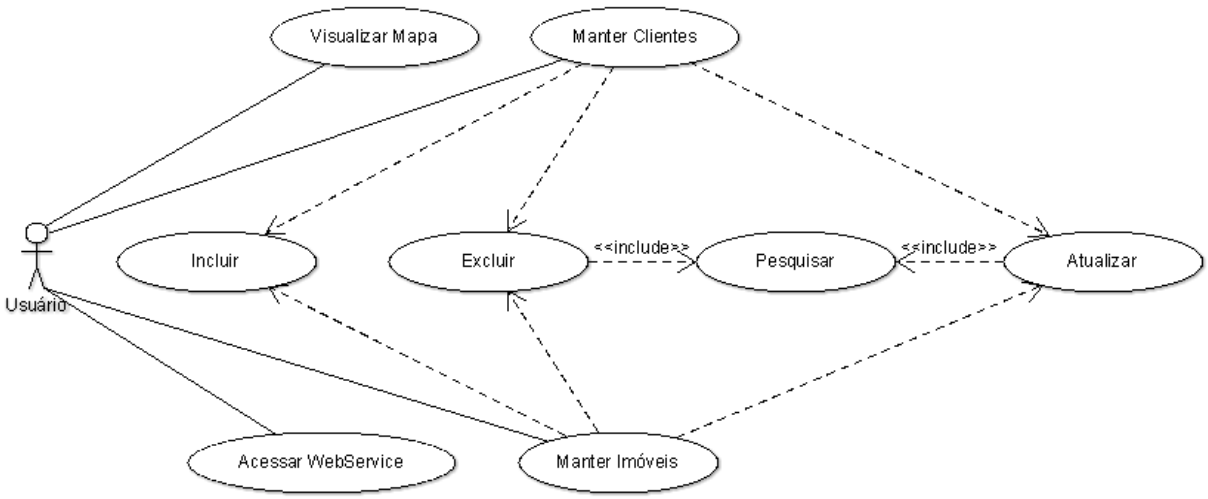


**Figura 27 – Casos de uso.**

### 3.3.2 – Diagramas de classes

As classes utilizadas no desenvolvimento deste aplicativo foram separadas em pacotes, sendo eles:

A figura 28 mostra o diagrama de classes do pacote *beans.*

- ***Beans***
  - **Proprietário:** classe responsável por instanciar os clientes do aplicativo.
  - **Imovel:** classe responsável por instanciar os imóveis do aplicativo.
  - **TipoImovel:** classe responsável por instanciar os tipos dos imóveis.
  - **Cidade:** classe responsável por instanciar as cidades.
  - **Estado:** classe responsável por instanciar os estados.
  - **Pais:** classe responsável por instanciar os países.

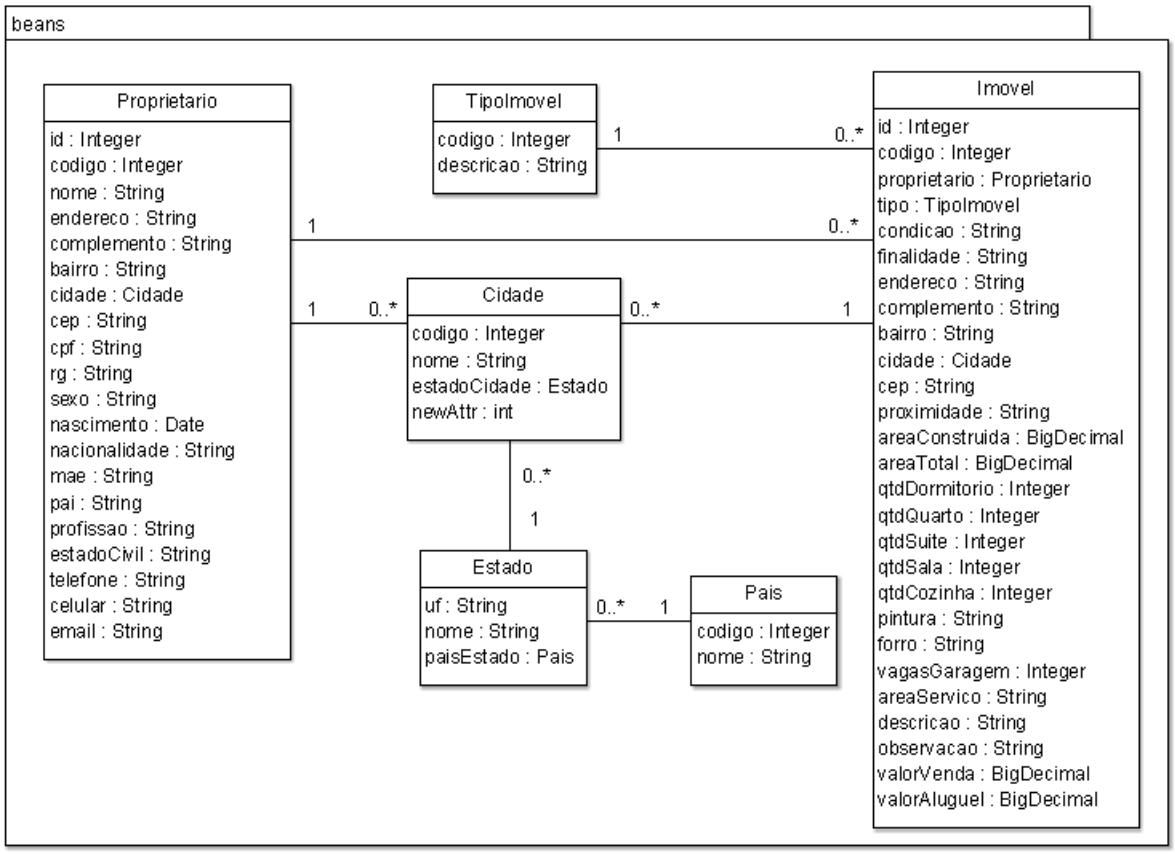


**Figura 28 – Diagrama de Classes - Pacote beans.**

A figura 29 mostra o diagrama de classes do pacote *daos*.

- ***Daos***
  - **Dao:** classe genérica responsável pela comunicação entre *beans* e banco de dados.
  - **ProprietarioDao:** classe responsável em armazenar os clientes no banco de dados.
  - **ImovelDao:** classe responsável em armazenar os imóveis no banco de dados.
  - **TipoImovelDao:** classe responsável em armazenar os tipos de imóveis no banco de dados.
  - **CidadeDao:** classe responsável em armazenar as cidades no banco de dados.
  - **EstadoDao:** classe responsável em armazenar os estados no banco de dados.
  - **PaisDao:** classe responsável em armazenar os países no banco de dados.

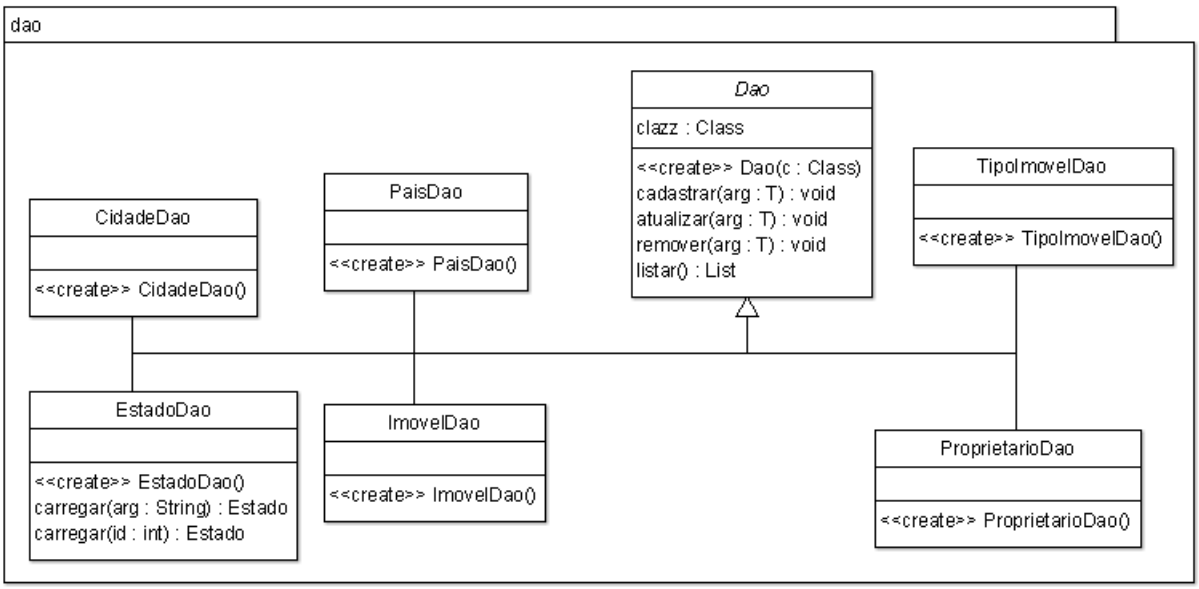


**Figura 29 – Diagrama de Classes - Pacote dao.**

A figura 30 mostra o pacote *manage beans*.

- ***Manage Beans*** (MB)
  - **ImovelMB**: classe responsável pela comunicação entre a página de JSF com os objetos da classe de imóveis.
  - **GMapsMB**: classe responsável por gerar o mapa com os marcadores de cada imóvel cadastrado no aplicativo.
  - **ProprietárioMB**: classe responsável pela comunicação entre a página de JSF com os objetos da classe de proprietários.

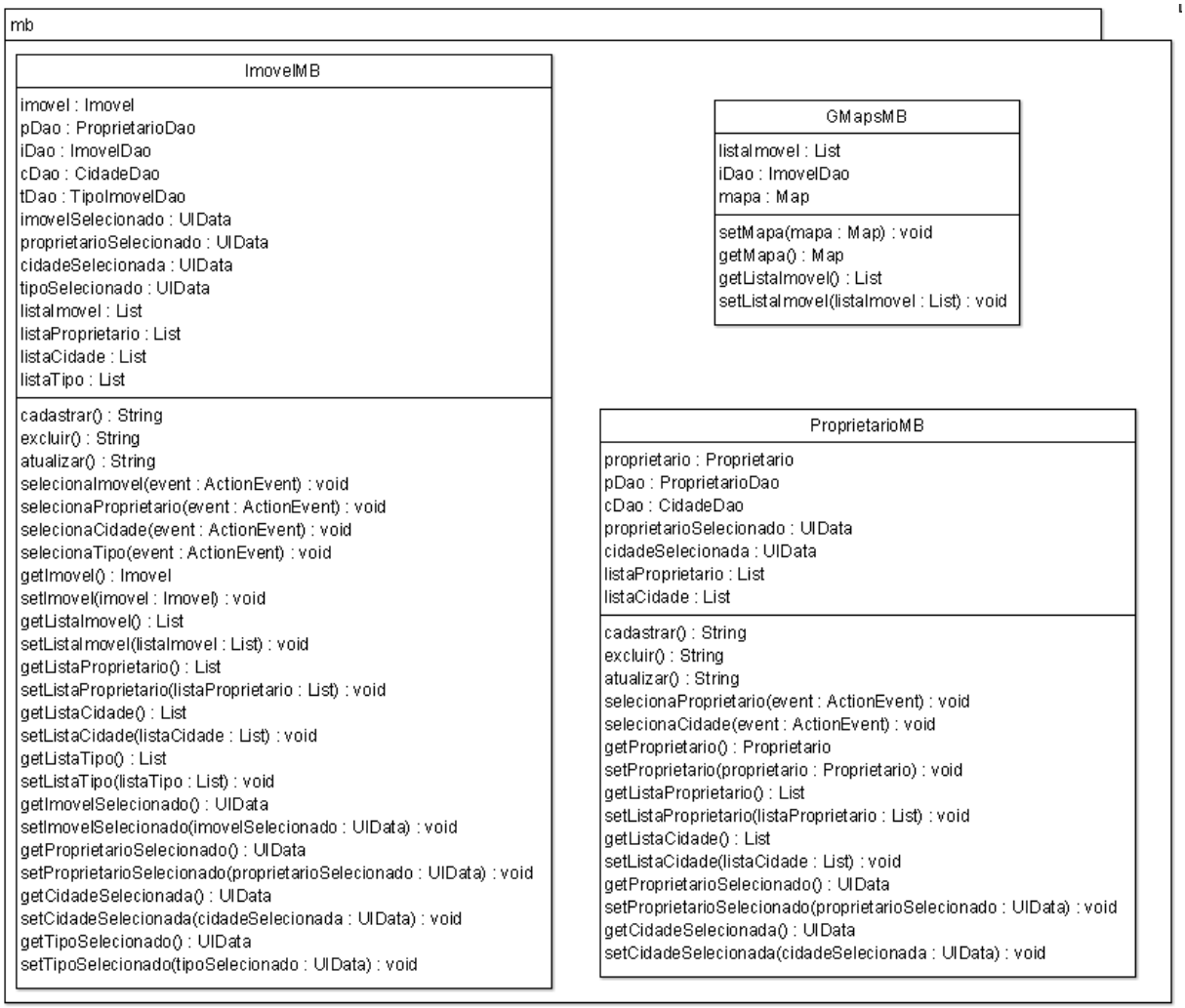


**Figura 30 – Diagrama de Classes - Pacote MB.**

A figura 31 mostra o diagrama de classe do pacote *Web Service*.

- ***Web Service*** (WS)
  - **ws:** classe responsável pelo *Web Service* que o aplicativo oferece, contendo cinco métodos essenciais para qualquer outro aplicativo usufruir deste serviço.

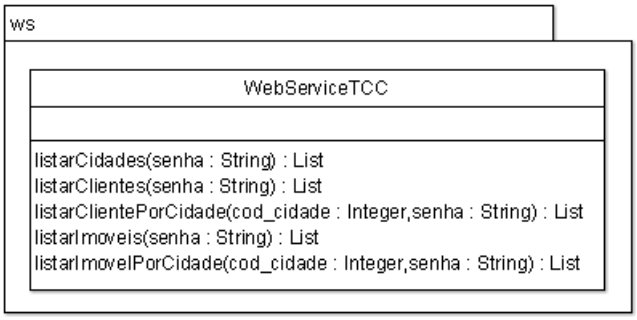


**Figura 31 – Diagrama de Classes - Pacote WS.**

### 3.3.3 – Diagrama de atividades

O diagrama de atividades visa representar o controle de fluxo das informações entre as atividades de um sistema. Na figura 32, é apresentado o diagrama de atividades do aplicativo, destacando todos os procedimentos necessários para o mapeamento de um imóvel cadastro.

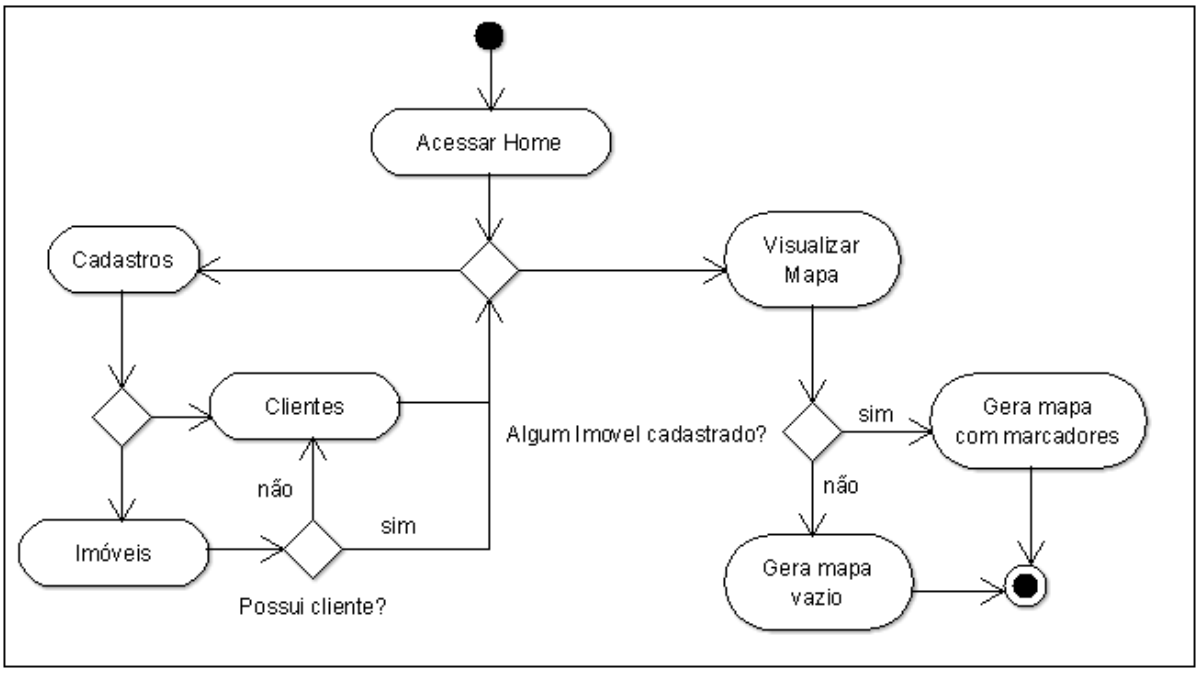


**Figura 32 – Diagrama de atividades.**

### 3.3.4 – Diagramas de seqüência

Os diagramas de seqüência representam a seqüência das ações ocorridas em um conjunto de classes, demonstrando como ocorre a troca de mensagens entre elas. Para cada caso de uso especificado, há um diagrama de seqüência, conforme detalhamento a seguir.

#### 3.3.4.1 – Manter clientes

Este diagrama de seqüência representado na figura 33 mostra as ações que podem ser executadas neste caso de uso, tais como: inserir, pesquisar, alterar e excluir cliente.

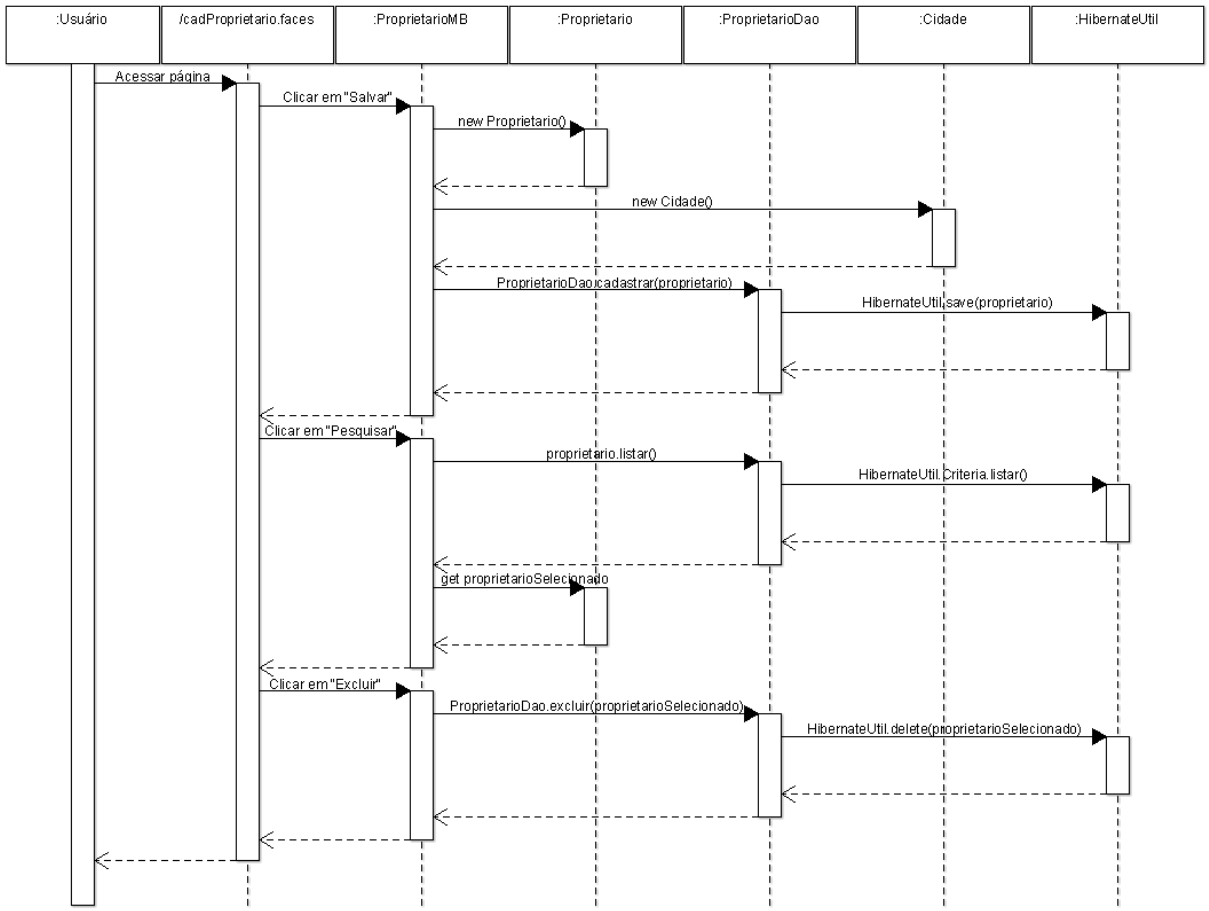


**Figura 33 – Diagrama de seqüência – Manter clientes.**

### 3.3.4.2 – Manter imóveis

O diagrama de seqüência que está representado na figura 34 mostra as ações que podem ser executadas neste caso de uso, tais como: inserir, pesquisar, alterar e excluir imóveis.

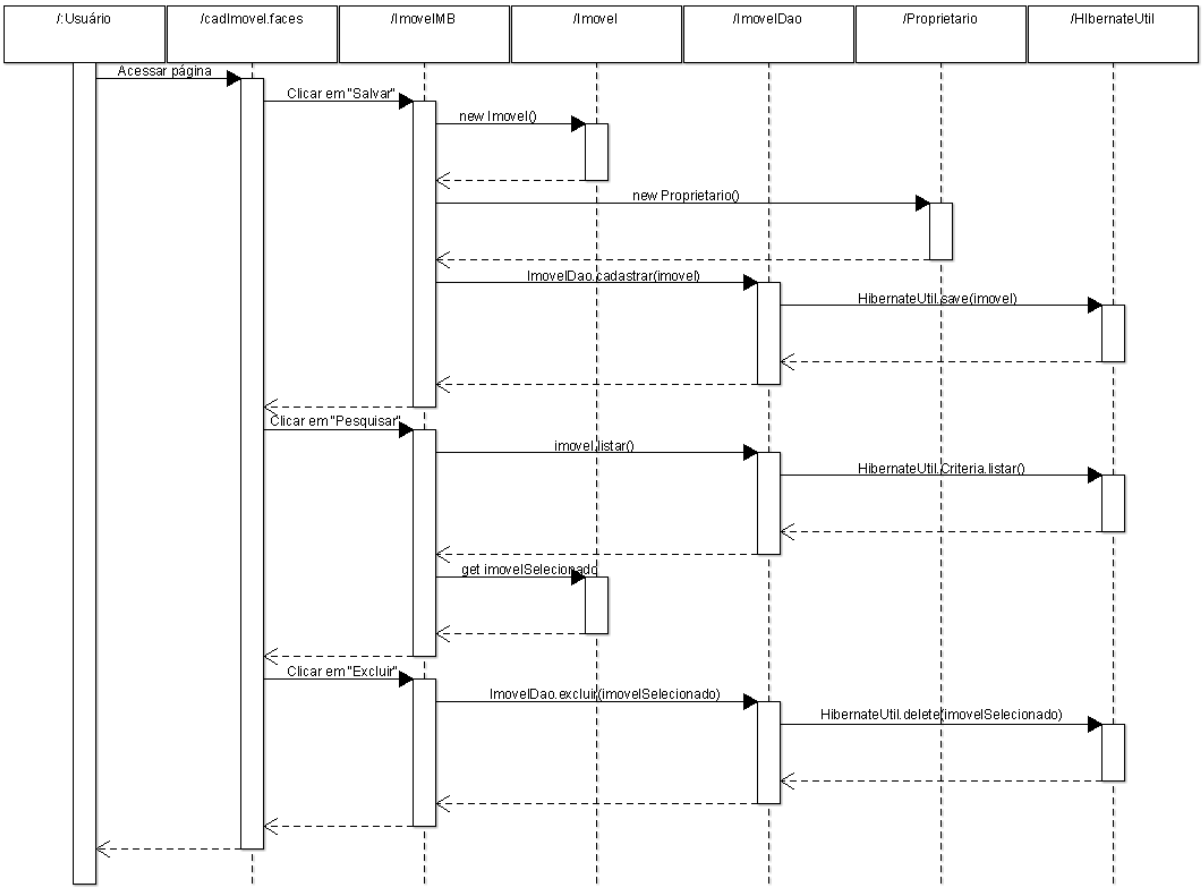


**Figura 34 – Diagrama de seqüência – Manter imóveis.**

### 3.3.4.3 – Visualizar mapa

Este diagrama demonstra as ações que são executadas quando o usuário entra na página de visualização do mapa, que são responsáveis em verificar se há imóveis cadastrados no aplicativo, para criar os marcadores para cada imóvel no seu local exato geograficamente. A figura 35 mostra o diagrama de seqüência para a visualização dos mapas.

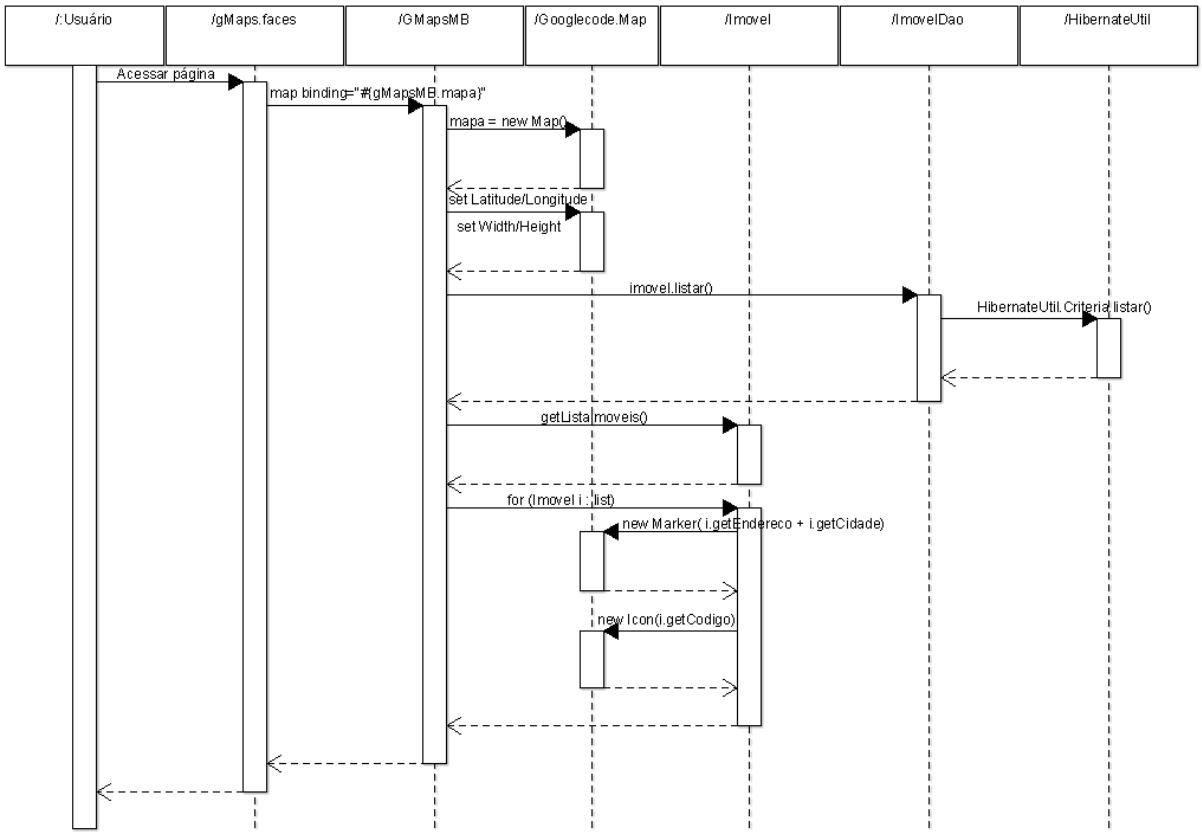


**Figura 35 – Diagrama de seqüência – Visualizar mapa.**

### 3.3.4.4 – Acessar *WebService*

A figura 36 mostra o diagrama de seqüência para acessar o *web service*. Este diagrama é responsável pelo serviço *web* disponibilizado no aplicativo. Ele possui um serviço com cinco métodos, que está disponível para qualquer sistema usufruir deste serviço.

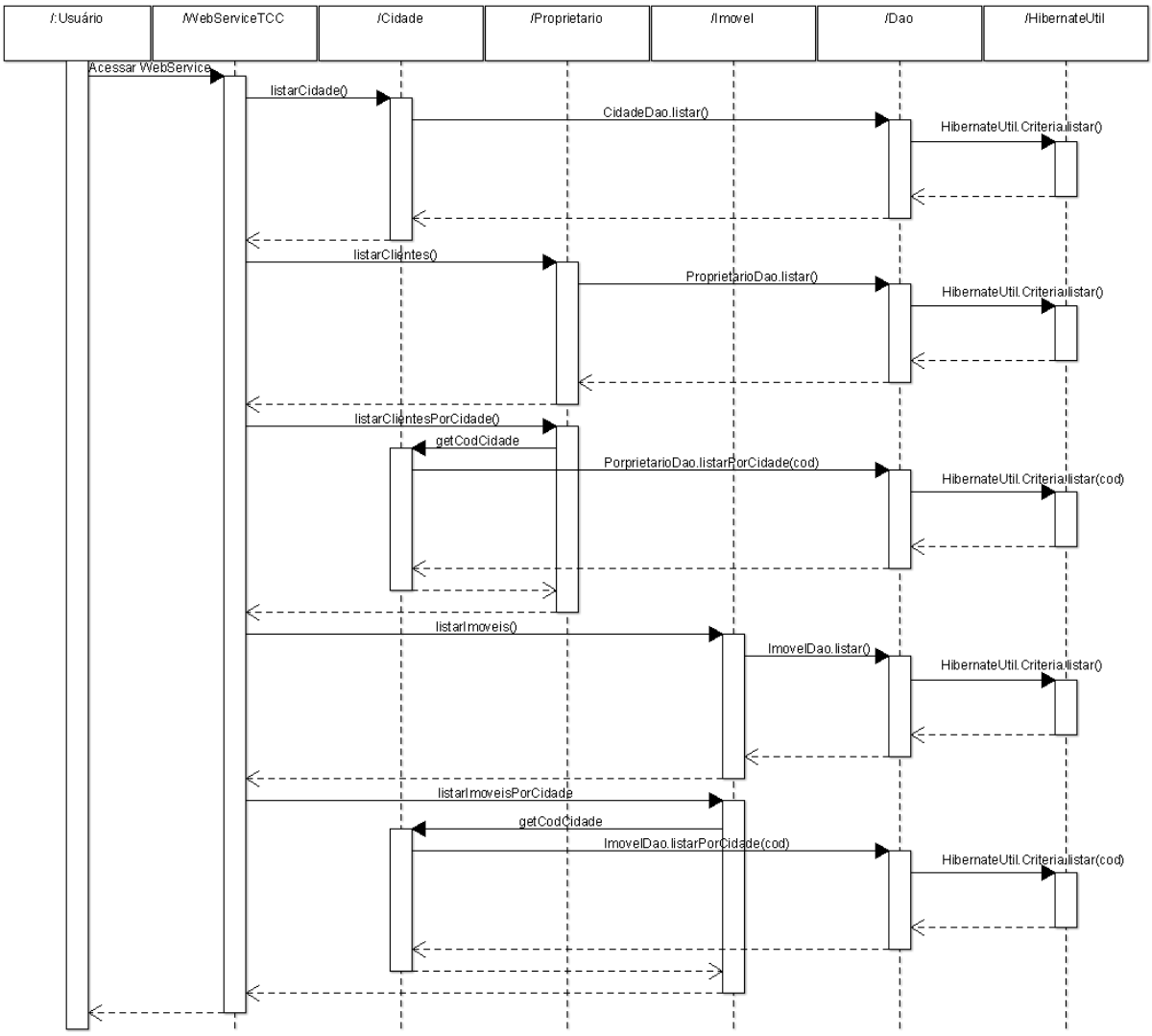


**Figura 36 – Diagrama de seqüência – Acessar *WebService*.**

## 3.4 - Implementação do Aplicativo

Serão apresentados os tópicos referentes à implementação do aplicativo desenvolvido neste trabalho, utilizando o ambiente de programação *NetBeans* 6.9

### 3.4.1 – Operacionalidade da implementação

O projeto desenvolvido neste trabalho é uma aplicação web com WebService embutido capaz de cadastrar clientes e imóveis, e disponibilizar a localização dos imóveis em um mapa. Para cada imóvel cadastrado é gerado dinamicamente um marcador no mapa contendo o código do imóvel, sendo capaz de localizar geograficamente o local exato do imóvel desejado.

Para a utilização da API do *Google Maps* é necessário fazer o *download* da mesma que se encontra no site da *Google Code*. Este API é um arquivo do tipo "*.jar*" que deve ser importado para o pacote de bibliotecas, fazendo com que ele seja integrado em seu projeto web.

A figura 37 apresenta a interface principal do aplicativo, onde o conteúdo da mesma pode ser dividida nas implementações dos casos de uso que seguem abaixo.

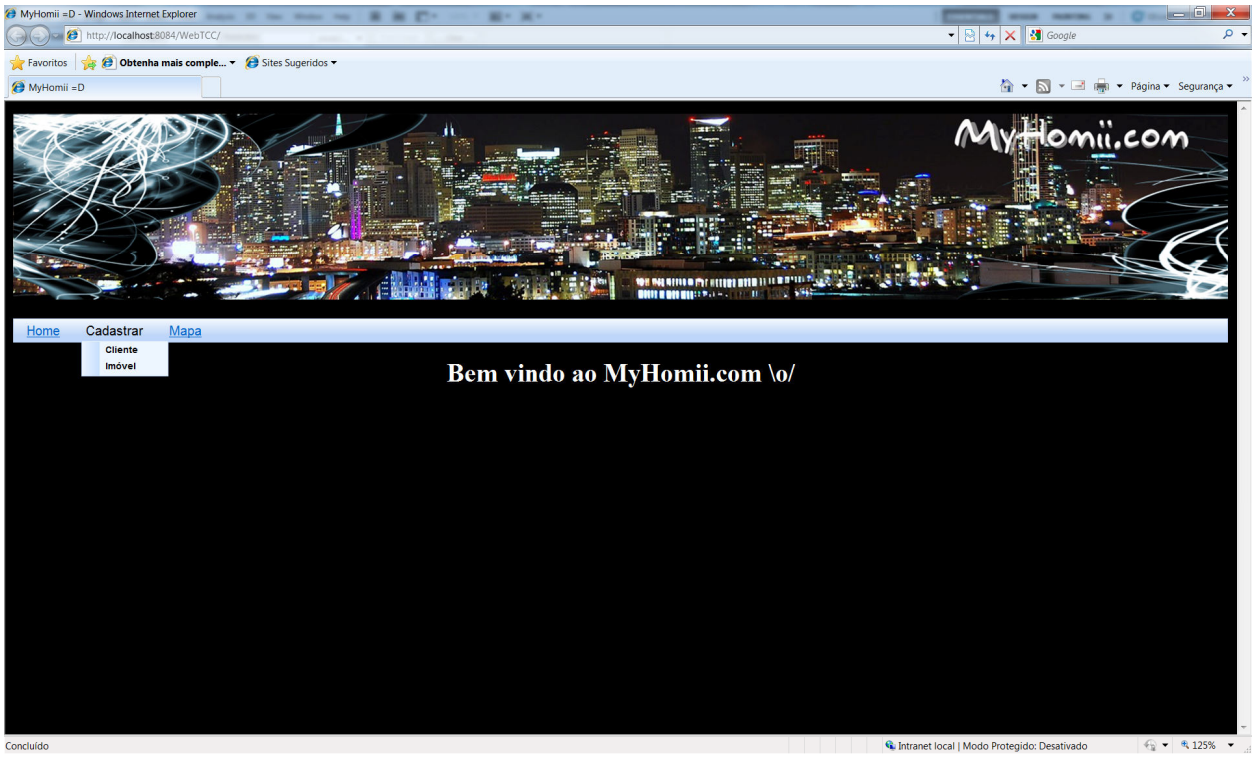


**Figura 37 – Pagina inicial do aplicativo *MyHomii.com*.**

### 3.4.1.1 – Implementação do caso de uso “Manter clientes”

O caso de uso “Manter clientes” ocorre quando um usuário deseja incluir, pesquisar, alterar ou excluir um cliente no aplicativo.

Para a inclusão de um cliente é necessário o preenchimento dos campos, sendo obrigatórios somente os campos: código, nome, endereço, cidade, RG e CPF. Para a escolha da cidade é necessário selecionar no ícone em forma de lupa na frente do campo “Cidade” para pesquisar a cidade desejada. Após esses passos, basta confirmar a inclusão através do botão “Salvar” como mostra a figura 38.

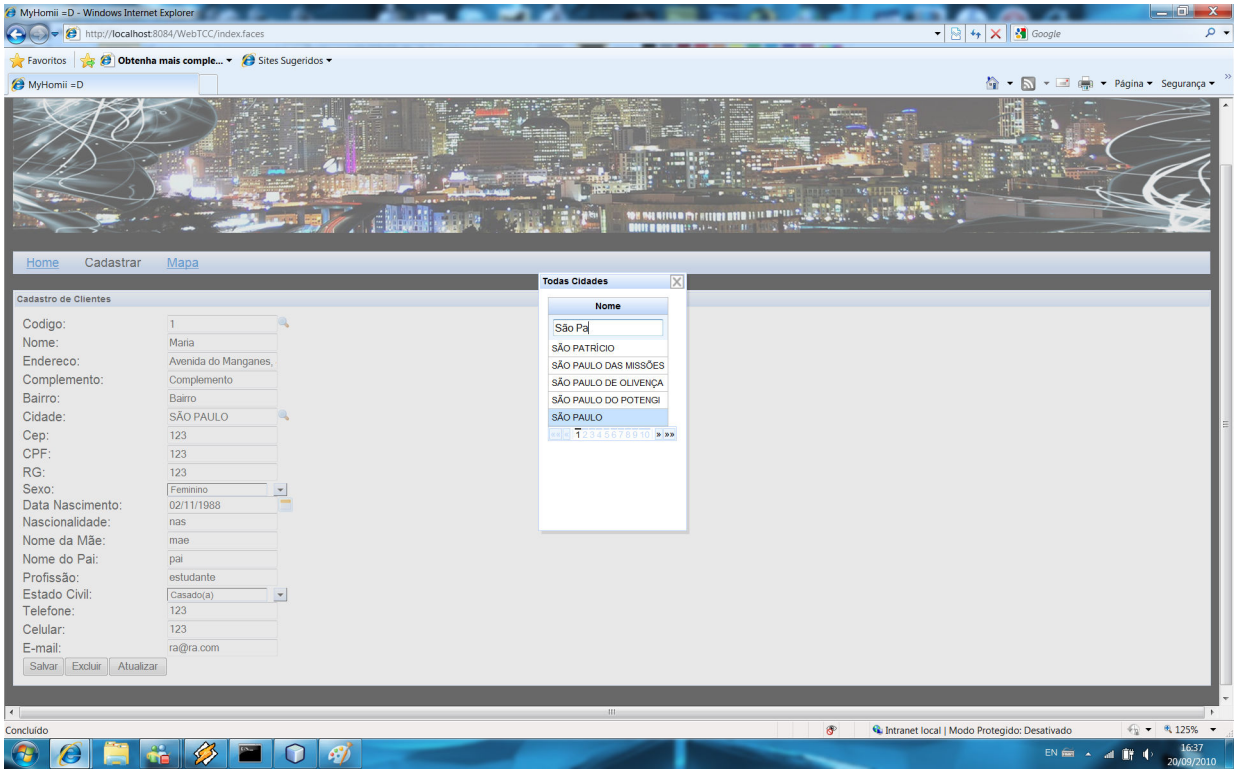


**Figura 38 – Selecionando uma cidade para incluir um cliente.**

Para excluir ou alterar um cliente, primeiro é necessário selecionar no ícone em forma de lupa na frente do campo “Código” para pesquisar o cliente desejado. Após a escolha do mesmo, será carregado todas as suas informações nos campos

correspondentes, agora basta escolher entre alterar os dados ou excluí-lo através dos botões “Atualizar” ou “Excluir” como mostra a figura 39.

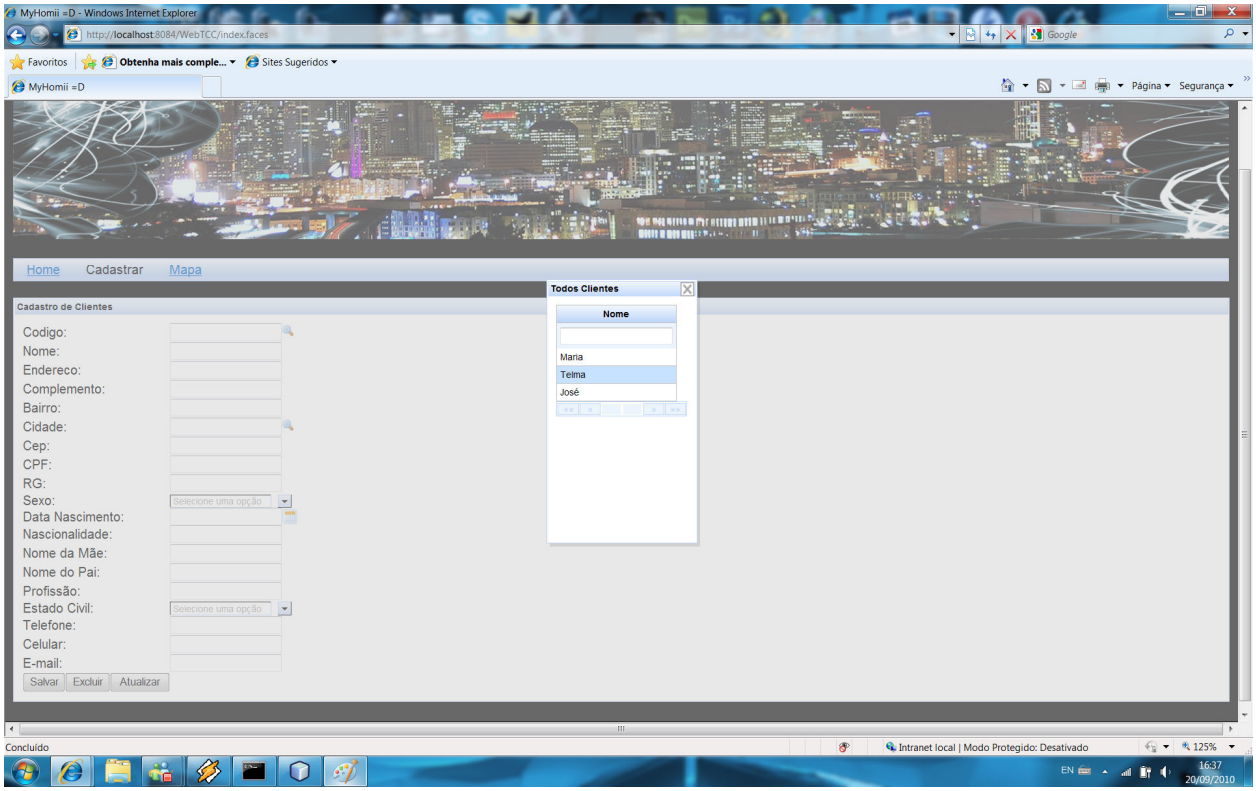


**Figura 39 – Selecionando um cliente para alterá-lo ou excluí-lo.**

### 3.4.1.2 – Implementação do caso de uso “Manter imóveis”

O caso de uso “Manter imóveis” ocorre quando um usuário deseja incluir, pesquisar, alterar ou excluir um imóvel no aplicativo.

Para a inclusão de um imóvel é necessário o preenchimento dos campos, sendo obrigatórios somente os campos: código, proprietário, tipo, endereço, cidade, valor de venda e/ou valor de aluguel. Para a escolha do proprietário é necessário selecionar no ícone em forma de lupa na frente do campo “Proprietário” para pesquisar o cliente desejado, o mesmo acontece para os campos “Tipo” e “Cidade”. Após esses passos, basta confirmar a inclusão através do botão “Salvar”.

Para excluir ou alterar um imóvel, primeiro é necessário selecionar no ícone em forma de lupa na frente do campo “Código” para pesquisar o imóvel desejado. Após a escolha do mesmo, será carregado todas as suas informações nos campos correspondentes, agora basta escolher entre alterar os dados ou excluí-lo através dos botões “Atualizar” ou “Excluir”, como mostra a figura 40.

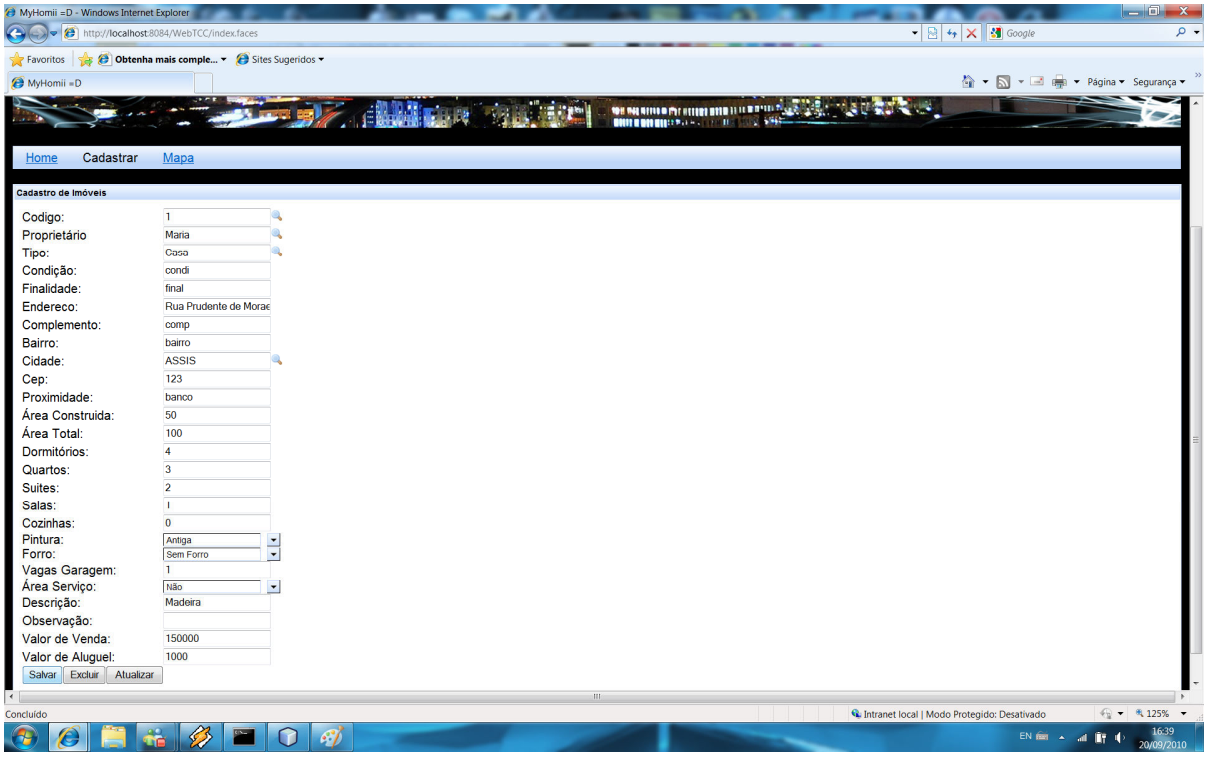


**Figura 40 – Página para a inclusão de um imóvel.**

### 3.4.1.3 – Implementação do caso de uso “Visualizar mapa”

O caso de uso “Visualizar mapa” ocorre quando um usuário seleciona o menu “Mapa” da página inicial. Neste momento a página cria o mapa fazendo um “*bind*” de componente no “*manage bean*” que busca todos os imóveis cadastrados no aplicativo e gera dinamicamente um marcador para cada imóvel em sua respectiva localização com seu respectivo código como mostra a figura 41.

Caso não haja nenhum imóvel cadastrado, o mapa é criado sem nenhum marcador.

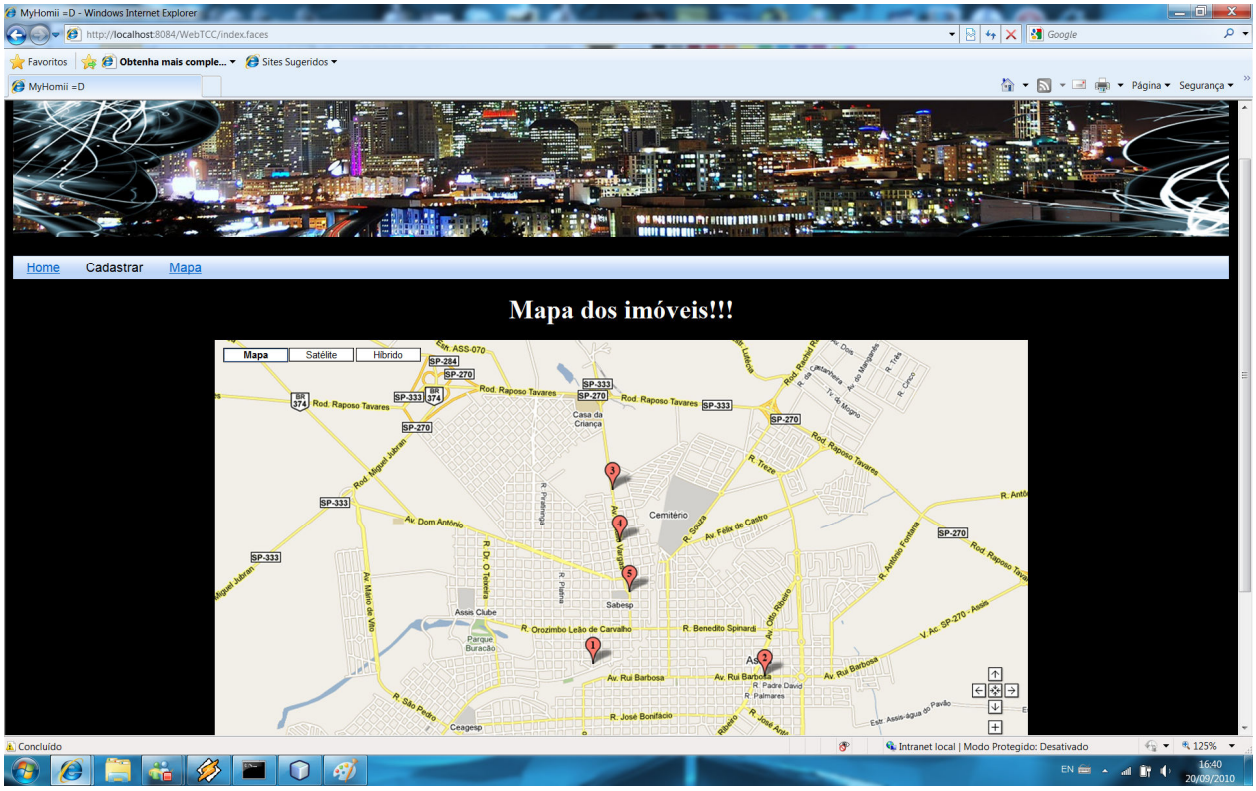


**Figura 41 – Página do mapa com os marcadores criados dinamicamente.**

### 3.4.1.4 – Implementação do caso de uso "Acessar *WebService*"

O caso de uso "Acessar *WebService*" ocorrerá quando outro aplicativo implementar os serviços oferecidos por este aplicativo. Sendo assim, isto torna o aplicativo escalável, pois futuramente pode-se desenvolver um aplicativo para celulares para acessar a mesma base de dados. A figura 42 mostra a interface de serviços contendo este aplicativo.

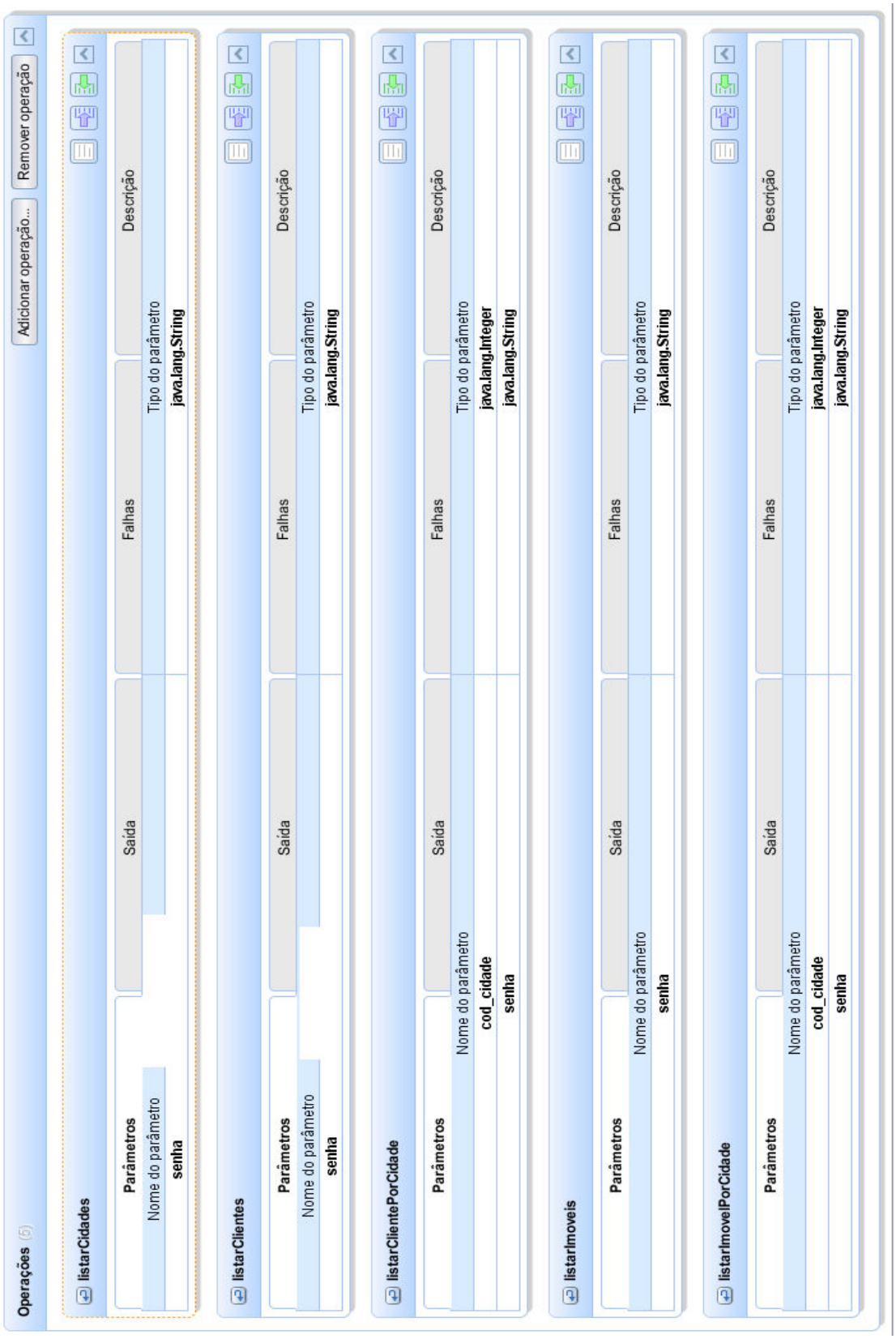


**Figura 42 – Serviços disponibilizados no *WebService* do aplicativo.**

# CAPÍTULO 4

# CONCLUSÃO

## 4.1 – Considerações finais

Este trabalho de conclusão de curso foi motivado pelo crescente interesse na utilização de aplicações *web*s com integrações com mapas, tanto nacionalmente como internacionalmente. O grande desafio foi aprender a desenvolver um sistema *web* com a API do *Google Maps*, utilizando os conceitos de *Servlets*, *Web Services*, programação Java *Web*.

Outro desafio foi aprender todo o mecanismo que envolve o princípio básico do mapeamento da API do *Google Maps*, para isso foi necessário realizar estudos sobre tecnologias para desenvolvimento *web*, criação de mapas e estudo de manipulação de dados espaciais.

As pesquisas realizadas sobre as ferramentas, *frameworks*, tecnologias e arquiteturas envolvidas para o desenvolvimento do aplicativo, são as principais contribuições deste trabalho. Contribui de forma direta para os alunos, como material de estudo e para empresas que desejam adotar o uso destes padrões de projeto em suas aplicações, melhorando a qualidade e a produtividade. Alem disso, os conhecimentos adquiridos para a realização deste trabalho foram de suma importância para o crescimento profissional e acadêmico. Sem dúvida esses conhecimentos serão fundamentais para a elaboração de outros projetos de pesquisa no futuro.

A aplicação web desenvolvida mostrou-se bastante eficaz para o gerenciamento dinâmico de localização dos imóveis cadastrados, e juntamente com o *web service* ele se torna um aplicativo escalonável que pode ser adicionado novos módulos futuramente.

## 4.2 – Trabalhos futuros

Como extensão deste trabalho pode-se estudar técnicas para deixar o mapa mais dinâmico, como por exemplo, implementar funções dinâmicas para clique do mouse e calcular e gerar rotas entre a atual localização do usuário e a localização do imóvel desejado.

Outra sugestão seria o estudo de realidade virtual para aplicar ao mapa, ou seja, o usuário na hora de escolher o imóvel teria acesso a um “passeio virtual” onde ele visualizaria todos os cômodos e suas atuais condições.

# REFERÊNCIAS BIBLIOGRÁFICAS

ALONSO, G; CASATI, F; KUNO, H. e MACHIRAJU, V.: **Web Services Concepts – Architectures and Applications**. Springer-Verlag, 2004.

ALVAREZ, M. A.: **Objetivos e usos do XML**. Disponível em: <http://www.criarweb.com/artigos/431.php>. Acesso em junho de 2010.

ALVES, G. D.: **J2ME – MAC 499**, Disponível em: <http://www.linux.ime.usp.br/~cef/mac499-04/monografias/givanildo/j2me.html>. Acesso em junho de 2010.

AMSDESIGN.: **B2C – Business to Consumer**. Disponível em: <http://www.amsdesign.com.br/B2C_Business_to_Consumer.asp>. Acesso em junho de 2010.

APACHE TOMCAT.: **Website oficial Apache Tomcat**. Disponível em: <http://tomcat.apache.org/>. Acesso em julho de 2010.

ARMSTRONG, E; BALL, J; BODOFF S.: **The J2EE Tutorial**. Sun Microsystems, 2005.

AZEVEDO, C.: **Imasters: Por Uma Internet Mais Criativa e Dinâmica**, Disponível em: <http://imasters.uol.com.br/artigo/7832/programacao/google_maps_api/>. Acesso em junho de 2010.

BARALE, R. F.: **Desenvolvimento de um sistema de Vendas na web**. Uberlândia, 2007.

BAUER, C. e KING, G.: **Hibernate in Action**, Manning Publications, 2004.
BERGSTEN, H.: **JavaServer Faces**, O'Reilly, 2004.

BONFIN JUNIOR, F.: **JSP – Java Server Pages – A tecnologia Java na Internet**. 1ed. São Paulo: Editora Érica, 2000.

BONFIN JUNIOR, F.: **JSP – Java Server Pages – A tecnologia Java na Internet**. 2ed. São Paulo: Editora Érica, 2002.

BROGDEN, B. e MINNICK, C.: **Desenvolvendo E-Commerce com Java, xml e jsp**. Pearson Education do Brasil, 2002.

BUHR, M. R. C. e VITORASSO C. L.: **UML e Java**. 2004.

CORESERVLETS.: **Servlets, JSP and JSF Tutorials**. Disponível em: <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>. Acesso em maio de 2010.

CUNHA, D.: **Web Services, SOAP e Aplicações Web**, Disponível em: <http://devedge-temp.mozilla.org/viewsource/2002/soap-overview/index_pt_br.html>. Acesso em junho de 2010.

DEITEL, H.M. e DEITEL, P.J.: **Java, como programar**. Pearson, Prentice Hall, 2005.

DRJT.: **Developer Resources for Java Technology**. Disponível em: <http://java.sun.com/>. Acesso em Abril 2010.

FARIA, N. A. de S.: **Suporte à edição cooperativa de Informação Geográfica em Ambiente WEB**. Dissertação (Mestrado em Informática) - Departamento de Informática, Escola de Engenharia, Universidade do Minho Braga, Portugal, 2006.

FISHER, M. e GREEN, D.: **The J2EE Tutorial for the Sun ONE Platform**. Disponível em: <http://java.sun.com/j2ee/1.3/docs/tutorial/doc/Front.html>. Acesso em abril de 2010.

FURTADO JUNIOR, M. B.: **Tutorial XML - Extensible Markup Language**. Universidade Federal do Rio de Janeiro, Disponível em: <http://www.gta.ufrj.br/grad/00_1/miguel/index.html>. Acesso em maio de 2010.

GEARY, D.: HORSMANN C.: **Core JavaServer Faces**, 2004.

GIGAFLOPS.: **Java Server Pages**. Disponível em: <http://gigaflops.tripod.com/page/lp/jsp/jsp.html >. Acesso em maio de 2010.

GOOGLE DEVELOPER GUIDE.: **Google Maps Java API Data**, Disponível em: <http://code.google.com/intl/pt-BR/apis/maps/documentation/mapsdata/developers_guide_java.html>. Acesso em junho de 2010.

GOOGLE MAPS. Disponível em: <http://maps.google.com.br/maps?ct=reset>. Acesso em junho de 2010.

GRANADOS, M. H.: **Arquitetura J2E.** Disponível em: <http://servidoresdeaplicaciones.wordpress.com/2009/03/09/arquitectura-j2ee/>. Acesso em junho de 2010.

JAVAFREE.: **Fórum Java Free.org**. Disponível em: <http://javafree.uol.com.br/wiki/Java>. Acesso em abril de 2010.

KORPER, S. e ELLIS, J.: **The e-commerce book - building the empire**. Academic press, 2000.

LIMERIA, J. L. S.. **Utilização de AJAX no desenvolvimento de sistemas Web**. Porto Alegre: 2006.

LIU, S.: **Getting Started with the Google Data Java Client Library**, Setembro 2007, Disponível em: <http://code.google.com/intl/pt-BR/apis/gdata/articles/java_client_lib.html>. Acesso em junho de 2010.

LOZANO F.: **Fernando Lozano Website.** Disponível em: <http://www.lozano.eti.br/>. Acesso em junho de 2010.

LUCCA, J. E.: **Integração de aplicativos com Web Services**. Universidade Federal de Santa Catarina, Florianópolis – SC, 2003.

RODRIGUES, M. C.: **Encapsulando a NavigationPlan Tool como Serviço Web**. IME – USP, 2008.

RUIZ, E. E. S.: **IBM – Programação Java Orientada a Objetos.** Disponível em: <http://dfm.ffclrp.usp.br/~evandro/ibm1030/intro_java/java_basics.html>. Acesso em abril de 2010.

SANCHEZ, O. V.: **Conhecendo Servlets**, 2004.

SEVERO, C. E. P.: **HSQLDB: um banco de dados livre escrito em Java**, Grupo de usuários Java, 2008.

SILVEIRA, P. E. A. e COSENTINO, R. A.: **CS-14 AED Java**. Caelum Ensino e Soluções em Java, 2009.

SILVEIRA, P. E. A. e COSENTINO, R. A.: **FJ-21 Java para desenvolvimento web**. Caelum Ensino e Soluções em Java, 2009.

SOUZA NETO, W. P.: **Usando Api do Google Maps para criar um mapa interativo**, Universidade Federal de Viçosa, 2009.

W3SCHOOLS.: **XML Tutorial**. Disponível em: <http://www.w3schools.com/xml/>. Acesso em junho de 2010.

XMLDESIGNER.: **Backgrounder da Tecnologia XML**, Disponível em: <http://msdn.microsoft.com/pt-br/library/8ktfywf4(VS.80).aspx>. Acesso em junho de 2010.