

**LUCIANE PIRES WERLANG  
JEFFERSON AMORIM DE OLIVEIRA**

**APLICAÇÃO DA MODULARIZAÇÃO NA ARQUITETURA E  
DESENVOLVIMENTO DE UM COMPONENTE DE PESQUISA BASEADO EM  
JAVA**

**Palhoça, 2006**

**LUCIANE PIRES WERLANG  
JEFFERSON AMORIM DE OLIVEIRA**

**APLICAÇÃO DA MODULARIZAÇÃO NA ARQUITETURA E  
DESENVOLVIMENTO DE UM COMPONENTE DE PESQUISA BASEADO EM  
JAVA**

Trabalho de Conclusão apresentado ao Curso de  
Ciência da Computação, como requisito à  
obtenção do título de Bacharel em Ciência da  
Computação.

Universidade do Sul de Santa Catarina

**Orientador Prof. M. Eng. Osmar de Oliveira Braz Junior**

**Palhoça, 2006**

**LUCIANE PIRES WERLANG  
JEFFERSON AMORIM DE OLIVEIRA**

**APLICAÇÃO DA MODULARIZAÇÃO NA ARQUITETURA E  
DESENVOLVIMENTO DE UM COMPONENTE DE PESQUISA BASEADO EM  
JAVA**

Trabalho de Conclusão apresentado ao Curso de  
Ciência da Computação, como requisito à  
obtenção do título de Bacharel em Ciência da  
Computação.

Universidade do Sul de Santa Catarina

Palhoça, 05 de Dezembro de 2006.

---

Prof. M. Eng. Osmar de Oliveira Braz Junior  
Universidade do Sul de Santa Catarina

---

Prof<sup>a</sup>. Msc. Vera Rejane N. Schuhmacher  
Universidade do Sul de Santa Catarina

---

Msc. Cristina Fogaça Soares Teixeira  
Convidado

## **DEDICATÓRIA**

A meu querido esposo Rodrigo, pelo apoio e ajuda em todos os momentos da minha vida.

Luciane Pires Werlang

Aos meus pais, Laurete e Nilton, pelo seu amor, carinho e confiança, fontes de sabedoria, exemplos de vida e integridade, sempre me apoiaram em todos os momentos da minha vida.

Jefferson Amorim de Oliveira

## **AGRADECIMENTOS**

Agradecemos, primeiramente, a Deus, pela vida e por iluminar nosso caminho.

Aos nossos pais, pelo exemplo de vida. Aos nossos irmãos, pelo apoio e carinho. Ao professor Osmar de Oliveira Braz Junior, nosso orientador, pelas contribuições, estímulo, atenção e orientação que nunca faltaram.

A todos os professores da UNISUL, que contribuíram para o nosso crescimento pessoal.

Aos amigos, pelos momentos de distração, alegria e ajuda.

Vocês foram essenciais!

## RESUMO

Este trabalho visa a aplicação da Ciência da Computação para análise, projeto e implementação de um sistema computacional. A necessidade de compreender as técnicas de modularização e componentização, assim como aplicar os conceitos da engenharia de *software* para elaborar uma arquitetura de sistema foram fundamentais para obter reuso e redução nos custos de manutenção e desenvolvimento de *software*. Além disso, utilizar uma metodologia de desenvolvimento que de suporte ao reuso, ao desenvolvimento de componentes e que permita que a mesma seja customizada. No trabalho foi utilizado o IBM *Rational Unified Process* (RUP). Essa metodologia foi customizada e serviu para gerenciar todo o processo de desenvolvimento do *software* e do componente e dessa forma permitir o acompanhamento de cada momento do desenvolvimento para evitar pular fases que poderiam pôr em risco o restante do projeto. A aplicação dessas técnicas, em conjunto com o processo de desenvolvimento, resultou em um componente de pesquisa totalmente reutilizável e também em um sistema de manutenção com pouca redundância e com manutenção facilitada. No entanto, para validar a aplicação dessas técnicas foram feitas comparações entre o sistema com a funcionalidade de pesquisa implementadas internamente e o mesmo sistema utilizando o componente para realizar as pesquisas. Os critérios utilizados foram código-fonte, tamanho da aplicação, tempo de desenvolvimento, coesão, acoplamento e reutilização. Dessa forma, com os resultados obtidos foram sugeridas algumas recomendações de uso.

**Palavras-chave:** Modularização, Componentização, Desenvolvimento de *Software*, Reuso, Engenharia de *Software*, Arquitetura de *Software*, IBM *Rational Unified Process*.

## **ABSTRACT**

This work aims at the application of the Computer Science for analysis, project and implementation of a computational system. The necessity to understand the techniques of creation of modules and development of components, as well as applying the concepts of the software engineering to elaborate a system architecture had been basic to get reuses and reduction in the maintenance costs and development of software. Moreover, to use a development methodology that of support to reuse, to the development of components and that it allows that the same one is adaptable. In the work IBM Rational Unified Process was used (RUP). This methodology was adaptable and served to manage all the process of development of software and the component and of this form to allow the accompaniment of each moment of the development to prevent to jump phases that could at risk put the remain of the project. The application of these techniques, in set with the development process, also resulted in a component of totality reusable research and in a system of maintenance with little redundancy and facilitated maintenance. However, to validate the application of these techniques had been made comparisons between the system with the research functionality implemented internally and the same system using the component to carry through the research. The used criteria had been code-source, size of the application, time of development, cohesion, coupling and reuse. Of this form, with the gotten results some recommendations of use had been suggested.

**Key words:** Creation of Modules, Development of Components, Development of Software, Reuse, Engineering of Software, Architecture of Software, IBM Rational Unified Process.

## LISTA DE FIGURAS

Figura 1. Proposta de Solução .....	20
Figura 2. Elicitação de Requisitos Arquiteturais .....	33
Figura 3. Princípios da Orientação a Objetos .....	39
Figura 4. Exemplo de Herança .....	41
Figura 5. Fases da Metodologia RUP .....	44
Figura 6. Representação de Componente em UML .....	46
Figura 7. Componentes e Interfaces .....	48
Figura 8. Estereótipos e Valores atribuídos .....	50
Figura 9. Interação entre um <i>Framework</i> e a Aplicação .....	52
Figura 10. Workflow Fase de Concepção .....	56
Figura 11. Workflow Fase de Elaboração .....	61
Figura 12. Workflow da Tarefa Projetar Componentes .....	66
Figura 13. Workflow Fase de Construção .....	70
Figura 14. Workflow da Tarefa Implementar Componentes .....	72
Figura 15. Workflow Fase de Transição .....	77
Figura 16. Cenário Tecnológico .....	82
Figura 17. Tela de Consulta .....	86



Figura 18. Tela de Cadastro.....	86
Figura 19. Tela de Edição.....	87
Figura 20. Tela de Consulta Cidade para o Cliente.....	88
Figura 21. Documento XML para Construção da Pesquisa .....	91
Figura 22. DTD de Validação para as Pesquisas.....	92
Figura 23. Dados da Conexão ao Banco de Dados no XML.....	92
Figura 24. Diagrama de Classes do Componente.....	94
Figura 25. Pacotes do Componente .....	95
Figura 26. TLD com os Parâmetros da <i>Taglib</i> .....	96
Figura 27. Código JSP utilizando a <i>Taglib</i> sem o Parâmetro “img” .....	97
Figura 28. Página demonstrando a <i>Taglib</i> sem o Parâmetro “img” .....	97
Figura 29. Código JSP utilizando a <i>Taglib</i> com o Parâmetro “img” .....	98
Figura 30. Página demonstrando a <i>Taglib</i> com o Parâmetro “img” .....	98
Figura 31. Trecho gerado pela <i>Taglib</i> .....	99
Figura 32. Página de Consulta.....	100
Figura 33. Localização do JAR e respectivos Arquivos de Configuração .....	101
Figura 34. Demonstração do Diagrama Entidade e Relacionamento.....	103
Figura 35. Diagrama de Interfaces do Sistema Manutenção com as Consultas Internas .....	104
Figura 36. Diagrama de Interfaces Utilizando o Componente para fazer as Consultas .....	105

## **LISTA DE QUADROS**

Quadro 1. Descrição dos Elementos da Proposta de Solução .....	21
Quadro 2. Diferenças entre Classe e Componente .....	47
Quadro 3. Métrica de Linha de Código .....	106
Quadro 4. Métrica de Tamanho do Vocabulário .....	107

## LISTA DE SIGLAS

DLL – *Dynamic-link library*

DTD – *Document Type Definition*

EXE – Extensão de arquivos que podem ser executados por computadores

Ferramentas CASE – Ferramentas *Computer Aided Software Engineering*

JSP – *Java Server Pages*

RUP – IBM *Rational Unified Process*

UML – *Unified Model Language*

XML – *Extensible Markup Language*

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>14</b>
1.1 Descrição do Problema.....	16
1.2 Objetivos.....	16
1.2.1 Objetivo Geral .....	17
1.2.2 Objetivos Específicos .....	17
1.3 Justificativa para o desenvolvimento do projeto .....	18
1.4 Proposta de Solução .....	19
1.5 Delimitação.....	21
1.6 Metodologia Científica Aplicada ao Trabalho .....	22
1.7 Estrutura do Trabalho .....	23
<b>2 REVISÃO BIBLIOGRÁFICA .....</b>	<b>24</b>
2.1 Engenharia de <i>Software</i> .....	24
2.1.1 Engenharia de Requisitos .....	27
2.1.2 Requisitos de <i>Software</i> .....	29
2.2 Arquitetura de <i>Software</i> .....	31
2.2.1 Modularização .....	34
2.2.2 Componentização .....	37
2.2.3 Orientação a Objetos .....	38
2.3 Análise e Projeto.....	42
2.4 <i>Unified Modeling Language</i> .....	45
2.5 <i>Frameworks</i> .....	51

2.6 Considerações Finais .....	53
<b>3 MODELAGEM.....</b>	<b>54</b>
3.1 Concepção .....	55
3.2 Elaboração .....	60
3.3 Construção .....	69
3.4 Transição .....	76
<b>4 DESENVOLVIMENTO.....</b>	<b>81</b>
4.1 Ambiente de Desenvolvimento .....	81
4.2 Implementação do Sistema Manutenção .....	85
4.3 Implementação do Componente LJ .....	88
4.3.1 Criação do XML e DTD.....	89
4.3.2 Criação das Classes do Componente .....	93
4.3.3 Criação da Página para Consulta.....	95
<b>5 VALIDAÇÃO .....</b>	<b>102</b>
<b>6 CONCLUSÃO.....</b>	<b>109</b>
<b>REFERÊNCIAS .....</b>	<b>112</b>
<b>APÊNDICES .....</b>	<b>115</b>
APÊNDICE A – Artefato do RUP de Plano de Desenvolvimento de <i>Software</i> .....	116
APÊNDICE B – Artefato do RUP de Especificação Complementar .....	123
APÊNDICE C – Artefato do RUP de Especificação de Requisitos de <i>Software</i> .....	129
APÊNDICE D – Artefato do RUP de Especificação de Caso de Uso .....	137
APÊNDICE E – Artefato do RUP de Documento de Arquitetura de <i>Software</i> .....	152
APÊNDICE F – Artefato do RUP de Plano de Iteração.....	162
APÊNDICE G – Artefato do RUP de Especificação da Realização do Caso de Uso.....	187
APÊNDICE H – Artefato do RUP de Plano de Implementação .....	224
APÊNDICE I – Artefato do RUP de Plano de Teste.....	230

APÊNDICE J – Artefato do RUP de Glossário.....	237
--	-----

## 1 INTRODUÇÃO

O princípio de “Dividir para Conquistar” era pouco utilizado quando o *software* constituía uma pequena porção dos sistemas computacionais. Entretanto, esse princípio começou a ser considerado quando o *software* dominou os sistemas computacionais tornando-se complexo e com custos de desenvolvimento e manutenção elevados. Tais fatos levaram especialistas a buscar métodos para melhorar o processo de desenvolvimento, assim como definir uma arquitetura de *software* que suportasse a reutilização de código e complexidade dos sistemas (MENDES, 2002; PRESSMAN, 1995).

A arquitetura de *software* descreve o sistema de uma forma global, bem como o relacionamento entre subsistemas e componentes. Segundo Pressman (1995, p. 429), “a arquitetura de *software* deriva de um processo de divisão em partições que relaciona elementos de uma solução de *software* a partes de um problema do mundo real”. Nesse sentido, a arquitetura é um guia no processo de desenvolvimento, no qual múltiplos sistemas com funcionalidades distintas podem ser concebidos.

Para Mendes (2002, p. 6), “o processo de desenvolvimento de um sistema de *software* vai desde a concepção do sistema, quando os requisitos são elicitados e analisados, até a sua concreta implementação”. Assim, na fase inicial do processo, há o interesse de compreender a funcionalidade do sistema e na fase de implementação, onde decisões já foram

tomadas, uma única arquitetura é implementada. Entretanto, antes da implementação de uma arquitetura, a funcionalidade do sistema é particionada, isto é, o sistema passa pelo processo de modularização.

A modularização como parte da arquitetura de *software* parte do princípio de dividir um problema em pequenas partes, sendo que cada uma dessas partes será independente e responsável pela realização de uma etapa do problema. Segundo Sanches (2006, p.10), “para que haja uma boa modularização é necessário principalmente baixo acoplamento e alta coesão”. O baixo acoplamento é definido por uma independência lógica dos módulos e a comunicação controlada entre eles. A alta coesão caracteriza-se por cada módulo exercer uma atividade específica do sistema.

Os módulos encontrados no processo de modularização, podem se tornar componentes. Esta componentização, por sua vez, ocorre quando um ou mais módulos se tornam uma parte física e substituível de um sistema e com o qual o respectivo componente está em conformidade e disponibilizado através de um conjunto de interfaces. Isso resulta em reutilização de código, sistemas otimizados e pouca redundância (BOOCH, 2000; BEZERRA, 2002).

Da mesma forma, quando encontramos módulos que pertencem a um mesmo domínio de problema e com funcionalidades comuns a várias aplicações temos o que chamamos de *framework*. Tipicamente, um *framework* pode incluir programas de apoio, bibliotecas, linguagens de script e outros *softwares* para ajudar a desenvolver e reunir diferentes componentes em um projeto de *software* (STAA, 2000; HUSTED, 2004).



## 1.1 Descrição do Problema

Nos últimos anos percebeu-se um grande avanço nos negócios relacionados com a tecnologia da informação. Com essa evolução, o *software* tem ocupado cada vez mais espaço e se tornado um componente vital para quase todos os segmentos de negócio.

Com essa crescente expansão, os *softwares* tornaram-se cada vez maiores e mais complexos, aumentando seus custos de produção e manutenção. Aliado a isto, as excessivas mudanças políticas e econômicas exigem que os sistemas tenham a manutenção facilitada, a fim de reduzir o tempo de adaptação a estas mudanças.

Percebeu-se então que para o desenvolvimento dos sistemas, a relação custo-benefício não estava boa. A partir disso, foram buscados meios para reduzir os custos no desenvolvimento e manutenção de *softwares* e assim obter mais lucro que é essencial para o funcionamento das economias nacionais e internacionais (SOMMERVILLE, 2003).

## 1.2 Objetivos

Os objetivos são divididos em:

- Objetivo Geral
- Objetivos Específicos

### 1.2.1 Objetivo Geral

Mostrar através da modularização e da componentização, uma maneira de desenvolver sistemas reutilizáveis e otimizáveis que permitam a redução de custos de produção e manutenção.

### 1.2.2 Objetivos Específicos

Os objetivos específicos englobam:

- Aprofundar os conhecimentos de Engenharia de *Software*, principalmente na área de arquitetura de *software*.
- Conhecer as técnicas de Modularização, Componentização e *Framework* como parte da arquitetura de *software*.
- Compreender as técnicas de arquitetura, análise e projeto de *software* no processo de desenvolvimento de sistemas que venham a trazer uma maior reusabilidade e manutenibilidade de código.
- Construir um sistema que demonstre a Modularização como parte do processo de desenvolvimento.
- Desenvolver um componente que se aproxime de um *framework* para ser utilizado pelo sistema.
- Analisar as vantagens e desvantagens de utilizar a Modularização no desenvolvimento de *software*.

- Analisar as vantagens e desvantagens de utilizar um componente dentro de sistemas de *software*.
- Aplicar as técnicas da computação para definir a modelagem e desenvolvimento do sistema e do componente.

### 1.3 Justificativa para o desenvolvimento do projeto

Com o crescimento gradativo da complexidade dos sistemas de *software*, houve a necessidade de utilizar meios que viessem a reduzir os custos e facilitar o desenvolvimento e a manutenção dos sistemas. Além disso, buscou-se também melhorar a qualidade do produto, assim como redução de defeitos dos *softwares* oferecidos.

Segundo Sommerville (2003), com a modularização fica mais fácil lidar com a complexidade, remover as redundâncias, otimizar as interações e simplificar as interfaces. A organização do ambiente de trabalho é outro fator importante para que o desenvolvimento de *software* ocorra com o mínimo de perdas, o que significa que não poderão ocorrer falhas durante o particionamento do programa.

Além disso, a modularização abre espaço para que várias pessoas possam trabalhar, ao mesmo tempo, em diferentes partes do sistema, visando sempre resolver o problema como um todo. Isso possibilita que a implantação e entrega do produto tenham os prazos reduzidos.

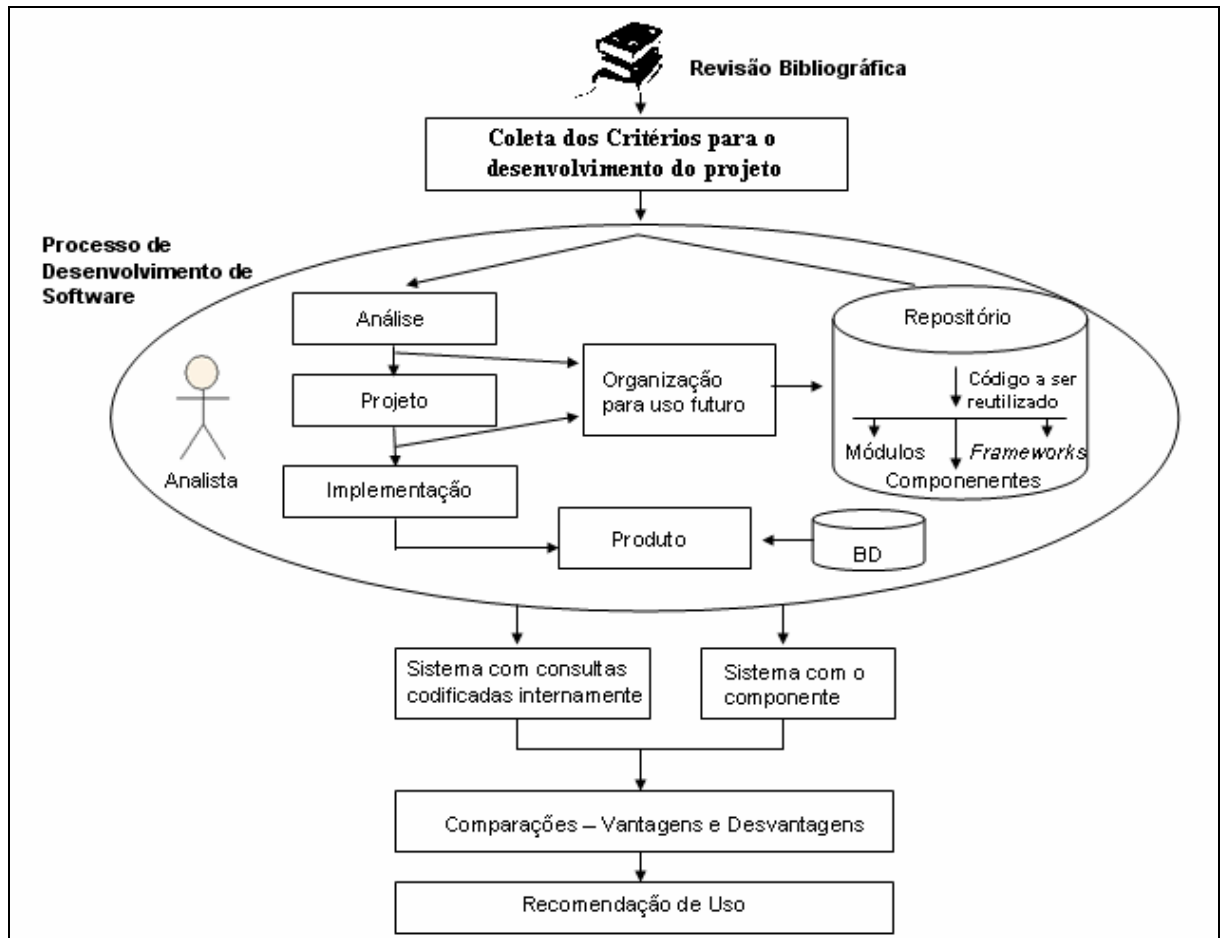
## 1.4 Proposta de Solução

A proposta de solução abrange uma pesquisa bibliográfica que é utilizada para coletar critérios. Esses critérios são usados como base para a adaptação de um processo de desenvolvimento que tem seu foco na reutilização de código.

Durante o processo de desenvolvimento trabalhar-se-á uma arquitetura que dê suporte a modularização, assim como a componentização. Com isso, as atividades de análise, projeto e implementação irão objetivar a organização para o futuro.

No sistema desenvolvido nesse processo existe uma funcionalidade específica de pesquisa. Essa funcionalidade é transformada em componente e o mesmo usado pelo sistema para realizar as pesquisas.

Para demonstrar as vantagens e desvantagens de utilizar componentes em um sistema de *software* é realizada a comparação entre o sistema com pesquisas implementadas internamente e o sistema utilizando o componente para executar essa funcionalidade. A partir disso é proposto uma recomendação de uso. A Figura 1 demonstra a proposta de solução:



**Figura 1. Proposta de Solução**

Fonte: Adaptado de Bezerra (2002).

A proposta de solução possui um conjunto de elementos que são fundamentais para o desenvolvimento de um produto que visa a reutilização e os mesmos são descritos no Quadro 1:

**Quadro 1. Descrição dos Elementos da Proposta de Solução**

Revisão bibliográfica	Conjunto de informações pesquisadas para desenvolver a monografia e atingir os objetivos propostos.
Analista	Especialista que trabalha no processo de desenvolvimento de <i>software</i> , definindo a arquitetura e desenvolvendo o sistema.
Processo de desenvolvimento de <i>software</i>	Processo de desenvolvimento que será adaptado para projetar um sistema que permita a reutilização de código.
Análise	Parte do desenvolvimento de <i>software</i> que busca analisar os requisitos do sistema.
Projeto	Parte do desenvolvimento de <i>software</i> que busca projetar, definindo os diagramas e a arquitetura do sistema.
Implementação	Parte do desenvolvimento de <i>software</i> que realiza a codificação do sistema, levando em consideração a análise dos requisitos e o projeto.
Repositório	Base de dados, no qual será armazenado todo o código considerado reutilizável, isto é, componentes, módulos, bibliotecas, <i>frameworks</i> , etc.

### 1.5 Delimitação

- Serão utilizados padrões de arquiteturas de *software* que possibilitam a reutilização de código e o uso da modularização.
- Os exemplos utilizados serão somente exibidos na linguagem de programação Java.
- Não é definida uma metodologia para o desenvolvimento do componente.
- Utilizará a metodologia de desenvolvimento IBM *Rational Unified Process* (RUP) customizado ao problema em questão.
- As instalações ou configurações do *software* utilizadas para a construção do sistema não serão discutidas.
- O sistema não irá se preocupar com a segurança de acesso, portanto, não será implementado cadastro de usuário e senha.

## 1.6 Metodologia Científica Aplicada ao Trabalho

A metodologia utilizada consiste em traçar um caminho que conduza à pesquisa e à investigação. Para Silva (2005), as classificações de pesquisa do ponto de vista de sua natureza, da forma de abordagem do problema, dos seus objetivos e dos procedimentos técnicos conceituam a metodologia científica utilizada em trabalhos.

Para o desenvolvimento deste trabalho foi adotado, considerando o ponto de vista de sua natureza, uma metodologia de pesquisa que segundo Silva (2005, p.20) “objetiva gerar conhecimentos para aplicação prática e dirigidos à solução de problemas específicos”.

Na abordagem do problema a pesquisa é qualitativa, isto é, o nível de informação coletada é fundamental e não sua estatística. Como ressalta Silva (2005, p.20) “[...]a interpretação dos fenômenos e a atribuição de seus significados são básicos no processo de pesquisa qualitativa. Não requer o uso de métodos e técnicas estatísticas[...]”.

Em relação aos objetivos a pesquisa é exploratória, pois busca tornar o problema mais explícito. Para isso, segundo Gil (*apud* SILVA, 2005, p.21) é necessário ter “[...]levantamento bibliográfico; entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; análise de exemplos que estimulam a compreensão”.

Por fim, quanto aos procedimentos técnicos a pesquisa é bibliográfica, isto é, segundo Gil (*apud* SILVA, 2005, p.21) “quando elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e atualmente com material disponibilizado na Internet”.

## 1.7 Estrutura do Trabalho

Esta monografia está organizada em seis capítulos. O presente capítulo apresentou o tema descrevendo o propósito da monografia e o problema de pesquisa, assim como os objetivos, a justificativa, a proposta de solução juntamente com suas delimitações e a metodologia utilizada.

No segundo capítulo encontra-se a revisão bibliográfica, que fundamenta as áreas de conhecimento envolvidas no trabalho: engenharia de *software*, arquitetura de *software* e o desenvolvimento de sistemas.

No terceiro capítulo é especificada a modelagem do sistema desenvolvido, que utiliza a metodologia RUP. Essa metodologia permite organizar o processo de desenvolvimento de *software* e separar por fases distintas a evolução do sistema objeto deste trabalho. Além disso, é descrito o ambiente tecnológico utilizado no trabalho.

No quarto capítulo, é descrito todo o processo de análise e desenvolvimento do componente com base na modelagem realizada anteriormente, assim como a demonstração de uma outra arquitetura para o sistema, com a utilização de um componente.

No quinto capítulo, é feita a validação do componente e sua arquitetura, apontando as vantagens e desvantagens de utilizá-lo em um sistema, além de uma recomendação de uso.

Finalmente, o sexto capítulo descreve a conclusão desta monografia, no qual estão descritos os resultados e as considerações finais.



## **2 REVISÃO BIBLIOGRÁFICA**

A revisão bibliográfica aborda assuntos que são fundamentais para o desenvolvimento da monografia. Inicialmente é feito um estudo de como a engenharia de *software* lida com a modularização e reutilização de código, além de demonstrar a importância da engenharia de requisitos no desenvolvimento de sistemas modulares. Na sequência são descritas a arquitetura de *software* e as suas técnicas, assim como conceitos de orientação a objeto e a importância da análise e projeto.

As áreas estudadas nessa revisão são todas direcionadas para a produção de *software* que proporcione redução nos custos de desenvolvimento, manutenção e também nos prazos de entrega do produto.

### **2.1 Engenharia de *Software***

Caracterizada por propor uma metodologia no desenvolvimento de *software*, a engenharia de *software* foi criada com base em conceitos de engenharia. O objetivo era

melhorar as técnicas que se utilizava para desenvolver sistemas computacionais (PRESSMAN, 1995).

Para Peters (2001), a grande vantagem de se iniciar projetos utilizando conceitos de engenharia de *software* são a simplificação e agilidade, devido a enorme disponibilidade de ferramentas e bibliotecas úteis para a criação de *software*. No lugar de construir novos sistemas de *software*, agora é possível comprar partes deles, ou adquirir pacotes completos, mas é importante saber qual *software* já está disponível e ao mesmo tempo criar e integrar o novo sistema. O reuso de *software* é atraente por causa da possibilidade de economizar tempo na resolução de um problema. A capacidade de reutilização de *software* é medida pela facilidade com que conceitos e objetos anteriormente adquiridos podem ser utilizados em novos contextos. Basicamente, a reutilização é uma combinação entre os componentes novos e os antigos. Assim, sempre que essa combinação for parcial ou bem-sucedida, o reuso é possível. Além disso, existe a vantagem de reduzir os custos do desenvolvimento.

A engenharia de *software* é uma disciplina que se ocupa dos aspectos para a produção do *software*. Esses aspectos incluem os processos técnicos de desenvolvimento de *software*, o gerenciamento de projetos de *software* e o desenvolvimento de ferramentas, métodos e procedimentos que dão apoio à produção de *software*. As ferramentas, métodos e procedimentos são considerados, por alguns autores, fundamentais para a construção de *software* com alta qualidade. Nesse sentido, as pessoas que trabalham nessa área sempre procuram encontrar soluções para os problemas, no entanto reconhecem que precisam trabalhar de acordo com as restrições organizacionais e financeiras, e assim estabelecem soluções que estejam dentro dessas restrições (SOMMERVILLE, 2003; PRESSMAN, 1995).

Um método de engenharia de *software*, segundo Sommerville (2003), é uma abordagem estruturada para o desenvolvimento de sistemas, com o objetivo de facilitar a produção de *software* de alta qualidade, apresentando uma boa relação custo-benefício.

Seguindo a mesma idéia, Pressman (1995) afirma que os métodos de engenharia de *software* proporcionam os detalhes de “como fazer” para construir o *software*.

Os métodos buscam identificar componentes funcionais básicos de um sistema, além de envolver um amplo conjunto de tarefas que incluem o planejamento e estimativa do projeto, análise de requisitos, projeto de estrutura de dados, arquitetura do sistema, codificação e teste. Esses métodos geralmente introduzem uma notação gráfica para demonstrar modelos que serão utilizados na especificação do sistema (SOMMERVILLE, 2003; PRESSMAN, 1995).

Nesse contexto, percebe-se que à medida que a capacidade de produzir *software* aumenta, cresce também sua complexidade. A complexidade dos sistemas é um desafio para os projetistas, além disso, o sistema tem de atender todos os requisitos, para que funcione conforme o desejo do cliente e a relação custo-benefício seja satisfatória. Sendo assim, a engenharia de *software* utiliza a modularização, que é uma técnica da arquitetura de *software*, como um método de auxílio para lidar com os sistemas complexos, tornando-os intelectualmente administráveis (PRESSMAN, 1995).

Além da complexidade, outro problema bastante comum nos dias de hoje é o tamanho dos sistemas. Para Sommerville (2003), quando nos deparamos com sistemas muito grandes a primeira atitude é identificar partes distintas do sistema e decompô-las em subsistemas e/ou módulos. No entanto, a flexibilidade, a complexidade e o reuso em um sistema ainda são as principais razões para subdividir os sistemas. Com técnicas de decomposição de *software* os sistemas se tornam confiáveis, fáceis de manter, permitindo a reusabilidade e a qualidade do *software*. Dentro desse contexto, Pressman (1995) destaca que os seres humanos desenvolvem uma abordagem natural para resolver seus problemas: se o problema é muito complicado, tendemos a subdividi-lo. Resolvemos cada um deles individualmente e esperamos que suas soluções possam ser combinadas para formar um todo.

### 2.1.1 Engenharia de Requisitos

Engenharia de Requisitos é a disciplina que procura sistematizar o processo de definição de requisitos para que o problema seja entendido de forma correta antes do comprometimento de uma solução. O grau de compreensibilidade, precisão e rigor da descrição do documento de requisitos de *software* tende a ser diretamente proporcional ao grau de qualidade do produto. Além disso, para que a definição de requisitos seja a mais eficaz possível, cabe aos engenheiros de *software* entender o ambiente no qual o *software* irá funcionar e escolher os modelos que melhor se encaixem no ambiente. (SOMMERVILLE, 2003; LEITE, 1994; PETERS, 2001).

Para o processo de engenharia de requisitos, Sommerville (2003) aponta as seguintes atividades: o estudo da viabilidade; a obtenção e análise de requisitos; a especificação de requisitos e sua documentação.

O estudo da viabilidade é o estágio inicial do processo de engenharia de requisitos que envolve uma descrição geral do sistema e de como será utilizado. Esse estudo resultará em um relatório que recomenda se vale ou não a pena realizar o processo de engenharia de requisitos. Além disso, poderá propor mudanças no enfoque, no orçamento e no cronograma, e também sugerir a inclusão de outros requisitos. É preciso dar um foco no objetivo do sistema, avaliando as informações coletadas e verificando se será possível chegar à esse objetivo (SOMMERVILLE, 2003; PRESSMAN, 1995; DENNIS, 2005).

Na seqüência é realizado o levantamento e a análise de requisitos. Esse estágio pode envolver muitas pessoas em uma organização como: desenvolvedores, analistas, clientes e usuários finais, os quais terão alguma influência direta ou indireta sobre os requisitos do sistema. Com o levantamento e análise dos requisitos é possível obter mais informações sobre

o domínio da aplicação, que serviços o sistema deve oferecer, desempenho exigido e se existe restrições (PRESSMAN, 1995).

O documento de requisitos, gerado do levantamento e análise, serve como meio de comunicação entre os diversos membros do desenvolvimento. Com o aumento cada vez maior na complexidade de sistemas e o desejo de aumentar a reusabilidade, tornam essa documentação muito importante. Quanto mais precisos, corretos, completos e consistentes os dados desses documentos maiores as chances de qualidade e reuso em futuras evoluções ou próximos desenvolvimentos (STAA, 2000; PRESSMAN, 1995).

Paralelamente à essas atividades, é necessário desenvolver o gerenciamento de requisitos. Essa atividade consiste em administrar as inevitáveis mudanças dos requisitos propostos que surgem, principalmente, quando são alteradas as prioridades do negócio, quando se identificam erros ou omissões nos requisitos ou quando novos requisitos são definidos. Gerenciamento de requisitos é executado por meio da implementação de rastreabilidade. Gerenciamento e rastreamento de requisitos são reconhecidos como importantes pré-requisitos para desenvolver *software* de alta qualidade.

Quando são propostas modificações, é preciso analisar e verificar o impacto dessas mudanças sobre outros requisitos. O gerenciamento de requisitos pode ter o apoio de ferramentas CASE, ou ainda recursos disponíveis em processadores de texto, planilhas de cálculo, etc. É preciso gerenciar também as mudanças de requisitos, pois assim todas as propostas serão tratadas de maneira consistente e as mudanças no documento de requisitos serão controladas (SOMMERVILLE, 2003).

### 2.1.2 Requisitos de *Software*

Os requisitos são as descrições de funções e de restrições de um *software*. O termo “requisito”, como argumenta Sommerville (2003), não é utilizado sempre com a mesma visão pela indústria de *software*. Em alguns casos é visto como declarações abstratas de alto nível, com linguagem natural, para uma função que o sistema irá oferecer ou uma restrição que irá ocorrer. Por outro ponto de vista, é uma definição detalhada de uma função – suas entradas, saídas e o próprio código - que o sistema deverá executar ou do serviço a ser prestado (SOMMERVILLE, 2003; STAA, 2000).

É importante fazer a separação entre esses dois níveis de descrição, pois alguns dos problemas que ocorrem durante o processo de engenharia de requisitos, surgem da falta nítida da separação desses níveis.

Ross et al. (*apud* STAA, 2000, p. 338) apresenta essa separação

“Requisitos não se referem apenas à funcionalidade desejada para um *software* (requisitos funcionais), mas também referem-se às questões não funcionais (por exemplo, desempenho, facilidade de uso), requisitos inversos (por exemplo, o módulo jamais suspenderá execução) e restrições (por exemplo, o módulo será redigido em Java)”.

Sommerville (2003, p.83, grifo nosso) define os requisitos funcionais e não funcionais da seguinte maneira:

“ 1. **Requisitos Funcionais:** São declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer.

2. **Requisitos Não Funcionais:** São restrições sobre os serviços ou as funções oferecidos pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros ”.

Os requisitos funcionais podem ser definidos como as funções ou atividades que o sistema realiza. Devem ser definidos claramente e relatados explicitamente. Podem ser elaborados a partir do relato das necessidades do cliente e/ou usuário, onde uma equipe de

projeto pode especificar efetivamente um sistema de informação, suas funções, desempenho, interfaces, restrições, etc., conforme as fases e subfases de uma metodologia de desenvolvimento de *software*. Eles são fundamentais para elaborar um sistema que atenda e satisfaça plenamente os anseios do cliente e da a equipe desenvolvedora do projeto.

Quando os requisitos são bem definidos e formalmente relatados evitam a alta manutenção de sistemas. Eles devem ser elencados por todos os envolvidos e principalmente com a concordância do cliente e/ou usuário.

Durante as fases de desenvolvimento, se ocorrerem problemas em função de uma má definição dos requisitos, esses devem ser corrigidos no respectivo documento, para uma adequada verificação futura.

Os requisitos não funcionais não são relacionados diretamente as funções específicas do sistema, mas relacionadas as propriedades que o sistema deve possuir, como confiabilidade, precisão, desempenho, segurança, exatidão, etc. (SOMMERVILLE, 2003; STAA, 2000). Muitos desses requisitos se referem ao sistema como um todo e não a partes individuais.

Por serem tão abrangentes, muitas vezes, os requisitos não funcionais são mais importantes que os requisitos funcionais individuais, pois se deixar de atender um requisito não funcional o sistema pode tornar-se inútil.

“Os requisitos não funcionais surgem conforme a necessidade dos usuários, em razão de restrições de orçamento, de políticas organizacionais, pela necessidade de interoperabilidade com outros sistemas de *software* ou hardware ou devido a fatores externos, como por exemplo regulamento de segurança e legislação sobre privacidade” (SOMMERVILLE, 2003, p.85).

Pelo fato de serem escritos refletindo os objetivos gerais do cliente, os requisitos são difíceis de ser verificados, gerando um problema comum para os desenvolvedores à medida que deixam o enfoque muito aberto à interpretação.

Para que esses problemas não ocorram, é preciso que os requisitos não funcionais fiquem expressos quantitativamente utilizando métricas, para serem mais verificáveis e,

assim, ser possível checar se o sistema atende ao objetivo exigido. No entanto, nem sempre essa especificação quantitativa é possível e fácil de ser definida.

Os requisitos funcionais e não funcionais devem ser diferenciados no documento que os descreve, para que não ocorram interpretações erradas entre os interessados no processo de desenvolvimento. Por outro lado, se forem definidos separadamente será difícil relacioná-los. Assim, o importante é encontrar um equilíbrio claro na definição e diferenciação dos requisitos.

## **2.2 Arquitetura de *Software***

A arquitetura de *software* surgiu para auxiliar o processo de desenvolvimento e demonstrar a estrutura global do sistema de *software*. O crescimento em termos de tamanho e complexidade dos sistemas levou pesquisadores e profissionais de engenharia de *software* a intensificar os estudos a respeito do assunto. Uma evidência é a grande quantidade de trabalhos, tais como linguagem de interface de módulos, arquiteturas específicas de domínio, linguagens para a descrição de arquiteturas e padrões de projeto. Considerada uma área relativamente nova dentro da engenharia de *software*, a arquitetura de *software* não havia provocado interesse dos pesquisadores até o final da década de 1980, quando Mary Shaw apontou a importância e a necessidade de considerar o nível organizacional ou arquitetural dos sistemas (MENDES, 2002).

Segundo Shaw et al. (*apud* VAROTO, 2002, p. 16) “arquitetura de *software* define o que é o sistema em termos de componentes computacionais e os relacionamentos entre estes componentes”, por sua vez, Bass et al. (*apud* VAROTO, 2002, p. 16) afirma



“Arquitetura de *software* são as estruturas que incluem componentes, suas propriedades externas e os relacionamentos entre eles, constituindo uma abstração do sistema. Esta abstração suprime detalhes de componentes que não afetam a forma como eles são usados ou como eles usam outros componentes, auxiliando o gerenciamento da complexidade”,

ou, ainda, para Jazayeri et al. (*apud* VAROTO, 2002, p. 16)

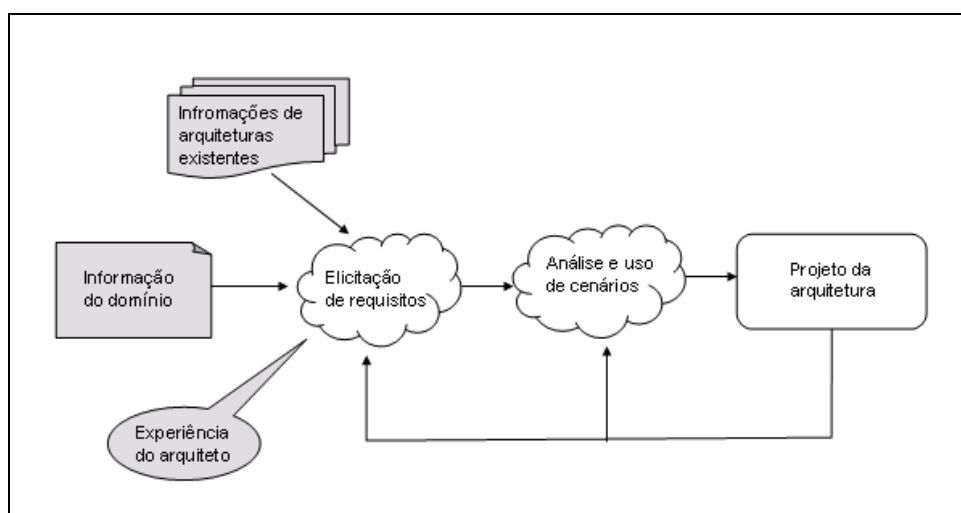
“A arquitetura de *software* é colocada como uma ferramenta para lidar com a complexidade do *software* e enfatizam que arquitetura deve satisfazer os requisitos funcionais e não funcionais do sistema, incrementando a definição de que arquitetura de *software* é o conjunto de componentes e seus relacionamentos. Portanto, é possível notar que a arquitetura de *software* é mais do que a descrição dos componentes que a compõem e do relacionamento entre eles. A arquitetura é a interface entre duas partes distintas: o problema de negócio e a solução técnica”.

A partir das definições acima citadas, percebe-se o quão importante e fundamental é a arquitetura de *software* e o quão essencial é o projeto arquitetural de um sistema dentro do processo de desenvolvimento. Além disso, Allen (2003, p. 62) menciona que “a arquitetura de *software* é importante, mas precisa ser reconfigurável para responder aos ambientes e as demandas em mutação”.

A arquitetura, considerada a estrutura geral de um sistema, pode conter subsistemas que interfaceiam com outros subsistemas e deve estar em um alto nível de abstração para que o sistema possa ser visto como um todo. A principal preocupação durante o processo de projeto de arquitetura, segundo Sommerville (2003, p.182) é “[...]estabelecer um *framework* estrutural básico[...]” que identifique os principais componentes do sistema e suas comunicações (ALLEN, 2003).

O processo de projeto de arquitetura é visto de diversos modos por diferentes projetistas e *stakeholders* (conhecedores e influenciadores do negócio). No entanto, o processo que será aproveitado dependerá da aplicação do conhecimento, da astúcia e da percepção clara e direta do arquiteto do sistema. Entretanto, existem atividades comuns a todos os processos de projeto de arquitetura de *software*, como a estruturação do sistema, a modelagem de controle e a decomposição modular. Assim, antes de termos um projeto de arquitetura formalizado, é necessário elicitar os requisitos arquiteturais (MENDES, 2002).

Durante o estágio de elicitação de requisitos, o arquiteto ou projetista faz uso de sua experiência para buscar identificar as peculiaridades do sistema a ser desenvolvido, além de utilizar as informações de domínio e estilos de arquiteturas diferentes. Contudo, temos a possibilidade de criar novos cenários, visando tanto a elicitação quanto a análise de requisitos, para então iniciar o projeto de arquitetura de *software* como mostrado na Figura 2 (SOMMERVILLE, 2003; MENDES, 2002).



**Figura 2. Elicitação de Requisitos Arquiteturais**

Fonte: Mendes, 2002, p. 37.

Projetar e documentar uma arquitetura de *software* para Bass et al (*apud* SOMMERVILLE, 2003, p. 182, grifo nosso) tem suas vantagens:

- “ 1. **Comunicação com os stakeholders:** a arquitetura é uma apresentação de alto nível do sistema, que pode ser utilizada como um ponto de discussão para uma gama de diferentes *stakeholders*.
2. **Análise de sistemas:** Tornar explícita a arquitetura de sistemas em um estágio inicial do desenvolvimento do sistema significa que alguma análise pode ser realizada. As decisões de projeto de arquitetura têm um profundo efeito sobre se o sistema pode ou não cumprir requisitos importantes, como desempenho, confiabilidade e facilidade de manutenção.
3. **Reutilização em larga escala:** a arquitetura de sistemas é uma descrição compacta e administrável de como um sistema é organizado e de como os componentes operam entre si. A arquitetura pode ser transferida por meio de sistemas com requisitos similares e, dessa maneira, pode fornecer apoio ao reuso de *software* em grande escala”.

Além disso, é importante salientar que para obter a arquitetura adequada julga-se necessário compreender os requisitos arquiteturais, que incluem os atributos do projeto e os requisitos não-funcionais. Os atributos do projeto podem ser vistos como princípios que

norteiam o processo de desenvolvimento de um *software*. Esses atributos são responsáveis por separar o sistema em várias partes, além de permitir que o projetista identifique diferentes aspectos do problema sem levar em consideração seus detalhes. Os requisitos não-funcionais descrevem o que o *software* não fará, mas como o fará. Isso significa que os requisitos não-funcionais abordam aspectos de qualidade importantes em um sistema de *software* que, se não levados em consideração, podem tornar o sistema inconsistente e de baixa qualidade. Exemplos desses requisitos são: o desempenho, a portabilidade, a manutenibilidade e a escalabilidade (MENDES, 2002).

### 2.2.1 Modularização

A modularização é considerada um atributo de projeto no processo de desenvolvimento de *software*. Caracterizada por decompor um sistema em partes, a modularização é indispensável quando o assunto é sistema complexo e/ou grande. A capacidade de decompor um sistema baseia-se na idéia de dividir o problema inicial em um conjunto de subproblemas e reaplicar tal procedimento em cada subproblema recursivamente. As partes resultantes da decomposição são chamados de módulos (MENDES, 2002).

Um módulo normalmente não é considerado um sistema independente. Segundo Peters (2001), o módulo é uma parte de programa logicamente separada. Cada módulo oculta decisões de projeto a respeito das características e conteúdos das estruturas de dados e exporta as operações de que o usuário necessita para utilizar o programa corretamente. Tudo isso leva a simplificação do projeto de *software*.

A modularização é o processo de reorganizar, de modo que as partes relacionadas sejam coletadas e consideradas um único módulo. É realizada manualmente, com a inspeção e a edição do código. Para modularizar é preciso identificar as relações entre componentes e examinar o que eles fazem. Feito isso, fica mais fácil remover as redundâncias, otimizar suas interações e simplificar suas interfaces.

Segundo Pressman (1995), um *software* monolítico ou um grande programa composto de um único módulo, é difícil de ser entendido pelo leitor. Assim, é mais fácil de resolver um problema complexo quando ele é dividido em partes administráveis, mas é necessário saber medir as conseqüências, e ponderar as divisões. À medida que o número de módulos cresce, o esforço (custo) associado à criação de interfaces também cresce. Devemos modularizar, mas com atenção.

Os módulos devem ser especificados e projetados de tal forma que suas informações sejam inacessíveis a outros módulos que não necessitem dela. Essa ocultação define e reforça as restrições de acesso, trazendo benefícios quando modificações são exigidas. Isso evita que essas modificações se propaguem a outros locais do sistema (PRESSMAN, 1995; DENNIS, 2005; STAA, 2000).

A modularidade, tanto de programas quanto de dados, possibilita que o projetista simplifique e reutilize os componentes de *software*. A ocultação de informações e a independência funcional oferecem os caminhos para conseguir efetiva modularidade. O agrupamento, que é facilitado pela modularização, produz exibições esquemáticas do *software* e permite uma maior compreensão do sistema, pois faz a separação da visão de alto nível e dos detalhes ocultos dos módulos.

Um projeto modular reduz a complexidade, facilita a mudança e resulta em uma implementação mais fácil ao estimular o desenvolvimento paralelo de partes do sistema.

Módulos independentes são mais fáceis de desenvolver, manter e testar. Desta forma, os efeitos secundários provocados por modificações são limitados.

A independência dos módulos é medida usando-se dois critérios: coesão e acoplamento. A coesão é uma medida da força funcional e depende do inter-relacionamento entre os elementos que constituem um módulo. Quanto mais forte for este inter-relacionamento, melhor será a coesão. O acoplamento é uma medida da interdependência e depende do volume de elementos que constituem a interface e da forma com que é estabelecida a interface. O acoplamento surge em função do relacionamento existente entre os módulos e é caracterizado pela passagem de controle entre eles (STAA, 2000).

Por sua vez, Pressman (1995, p. 441) estabelece que um “módulo coesivo executa uma única tarefa dentro do procedimento de *software*, exigindo pouca interação com procedimentos que são executados em outras partes de um sistema”. Dessa forma, um módulo coesivo deve fazer apenas uma coisa. Em relação ao acoplamento, o nível em um projeto de *software* deve ser o mais baixo possível.

Na modularização um módulo deve possuir uma elevada coesão para permitir a execução de uma tarefa procedimental distinta. Por outro lado, é necessário o baixo acoplamento, que para Pressman (1995) é possível utilizando um número mínimo de interfaces e informações que são trocadas por elas.

No entanto, deve-se considerar a baixa coesão de forma que o projeto possa ser modificado para conseguir maior independência funcional, já que o acoplamento depende da complexidade de interfaces entre os módulos. A simples conectividade entre os módulos resulta em um *software* que é mais fácil de entender e menos propenso a propagação de erros pelo sistema (PRESSMAN, 1995).

### 2.2.2 Componentização

Um componente é um conjunto de um ou mais módulos, formando um todo coerente e implementando uma funcionalidade bem definida. Componentes são incorporados ao programa sem sofrer alterações, além de serem mais abstratos do que classes e poder ser considerados provedores de serviços. Quando o sistema precisar desse serviço basta chamar o componente, sem se preocupar onde tal componente está sendo executado (STAA, 2000; BEZERRA, 2002; SOMMERVILLE, 2003; BOOCH, 2000).

Por sua vez, Peters (2001) caracteriza um componente como uma unidade de *software* testada para fins especiais. Por exemplo, uma classe Java, extensivamente testada, considerada confiável que seja útil, adaptável e reutilizável.

Assim, para visualizar um componente reutilizável como provedor de serviços é preciso enfatizar, segundo Sommerville (2003), duas importantes características:

- É uma entidade executável independente. O código-fonte não está disponível e não é compilado com outros componentes do sistema.
- Publicam suas interfaces e todas as interações são feitas por meio dessa interface. Sua interface é expressa e seu estado interno nunca é exposto.

A reusabilidade é uma característica importante de um componente. Ela permite que o componente, depois de ser projetado e implementado, possa ser reusado em muitos programas ou sistemas diferentes. Os componentes reutilizáveis são mais confiáveis do que novos componentes, já que foram experimentados e testados, reduzindo o número de falhas.

O processo ideal de desenvolvimento de componentes, segundo Sommerville (2003), deve ter como base a experiência, em que os componentes reutilizáveis são construídos a partir de componentes existentes e já utilizados. Utilizando o conhecimento

sobre problemas de reuso e as adaptações necessárias dos componentes, pode ser criada uma versão de componente mais genérica, portanto, mais reutilizável. Tornar um componente reutilizável consiste em fornecer uma interface genérica, com operações que apresentam as diferentes maneiras pelas quais o componente pode ser utilizado.

O principal objetivo do desenvolvimento baseado em componentes é permitir que os desenvolvedores usem mais de uma vez o código escrito em qualquer linguagem. Seu principal propulsor é a projeção de unidades de *software* individuais “conectáveis” – unidades que possam ser facilmente conectadas a uma aplicação para estender seu funcionamento.

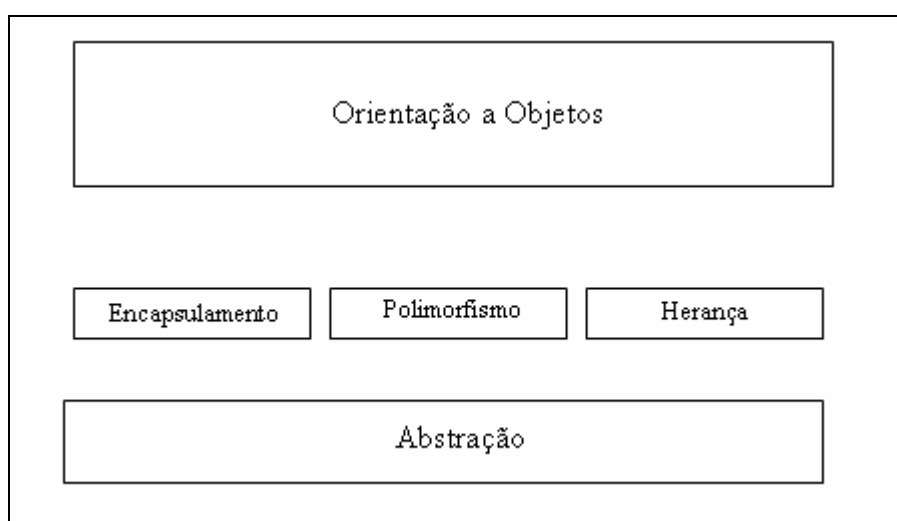
### 2.2.3 Orientação a Objetos

O termo orientação a objetos pressupõe uma organização de *software* em termos de coleção de objetos discretos incorporando estrutura e comportamento próprios. Esta abordagem de organização é essencialmente diferente do desenvolvimento tradicional de *software*, onde estruturas de dados e rotinas são desenvolvidas de forma apenas fracamente acopladas. Além disso, a orientação a objetos é edificada em cima dos conceitos de modularidade, abstração e ocultação de informações (PRESSMAN, 1995).

A abstração, segundo Mendes (2002), é a maneira mais apropriada para lidar com a complexidade. Através da abstração podemos identificar os aspectos importantes de um fenômeno e ignorar os detalhes. Pressman (1995) comenta que a abstração pode ser apresentada em vários níveis. Em um nível mais elevado a solução é declarada em termos amplos, usando a linguagem do ambiente do problema. Já em um nível inferior, a solução é definida de uma forma que possa ser diretamente implementada.

Na orientação a objetos, a abstração funciona como uma ferramenta, que possibilita compreender o sistema. Dessa forma, o sistema é dividido em componentes e os mesmos são visualizados em diferentes níveis de complexidade e detalhe.

Bezerra (2002) coloca que o paradigma da orientação a objetos é uma forma de abordar um problema. Alguns conceitos importantes como abstração, objeto, classes, encapsulamento, herança e polimorfismo devem ser considerados quando utilizamos esse paradigma no desenvolvimento de *software*, como mostra a Figura 3:



**Figura 3. Princípios da Orientação a Objetos**

Fonte: Bezerra, 2002, p. 09.

Um objeto é a abstração de uma entidade do mundo real que é mapeado para o domínio de *software*. Um objeto define os fatos. Na linguagem de um programador, um objeto é a instância de uma classe. Uma classe corresponde a descrição dos atributos e serviços comuns a um grupo de objetos, isto é, um molde a partir do qual os objetos são construídos. Uma classe define as regras. Além disso, as classes são utilizadas para agregar ações fortemente interdependentes e que encapsulam a informação de estado manipulada por estas ações.

Dentro deste contexto, o encapsulamento é a forma pela qual podemos restringir o acesso a operações que pertencem à classe, a fim de evitar que as mesmas sejam modificadas. Assim, a comunicação entre objetos será feita através de funções de acesso. O

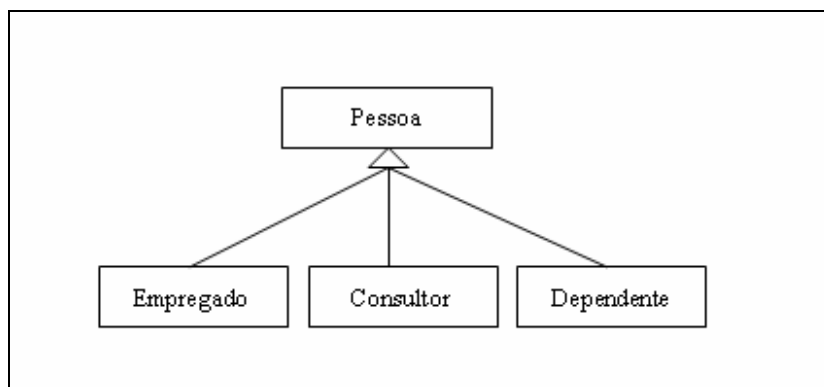


encapsulamento descreve uma maneira de organizar as informações em uma abstração de modo que possam ser usadas eficientemente em uma aplicação de *software*. O encapsulamento separa os fatores necessários a fim de usar o objeto dos fatos necessários para fazê-lo funcionar corretamente. Para usar um objeto, ele precisa expor sua finalidade e suas interfaces. Para fazer o objeto funcionar corretamente, ele precisa conter os dados e o comportamento que satisfaçam os serviços oferecidos pelas interfaces (PRESSMAN, 1995; STAA, 2000; PAGE-JONES, 2001; BEZERRA, 2002).

Como o encapsulamento restringe o acesso interno dos objetos, a ocultação de informações utiliza o encapsulamento para restringir a visibilidade externa de certos detalhes de informação, isto é, o observador externo tem pleno conhecimento do que o objeto pode fazer, mas não sabe como ele pode fazer ou como ele é constituído (PAGE-JONES, 2001).

Um objeto consegue restringir o acesso a suas informações através do encapsulamento e ocultação de informações. No entanto, um objeto também tem a capacidade de reter informações indefinidamente, inclusive durante os intervalos de ativação de suas operações.

A herança é outro conceito da orientação a objetos que define a relação através da qual uma classe especializa, refina ou particulariza propriedades de outra classe. Na herança, classes semelhantes são agrupadas em hierarquias. Cada nível de uma hierarquia pode ser visto como um nível de abstração. Assim, cada classe herda as características das classes dos níveis acima, o que facilita o compartilhamento de comportamento comum a essas classes e permite organizar, de forma clara, as diferenças e variações de uma determinada classe em particular. Na Figura 4, é ilustrado um exemplo de herança, onde as propriedades da superclasse Pessoa são herdadas pelas classes herdeiras Empregado, Consultor e Dependente (STAA, 2000).



**Figura 4. Exemplo de Herança**

Fonte: Adaptado de Staa (2000).

Uma característica associada à herança é o polimorfismo. O polimorfismo indica a capacidade de abstrair várias implementações diferentes em uma única interface, isto é, um objeto pode enviar a mesma mensagem para objetos semelhantes, mas que implementam a sua interface de formas diferentes.

Conforme Pressman (1995), para conseguir modularidade em um projeto de *software* é necessário: capacidade de decompor o problema em subproblemas; capacidade de composição, no qual os componentes (módulos), uma vez projetados e construídos, possam ser reutilizados em outros sistemas; compreensibilidade; continuidade e proteção. Os três últimos referem-se: a facilidade com que um componente pode ser entendido, sem referência a outras informações ou módulos; capacidade de fazer pequenas mudanças sem grandes consequências e a redução da propagação de efeitos colaterais se um erro ocorrer em um determinado módulo. Assim sendo, percebeu-se que a orientação a objetos realiza cada critério, acima relacionado, de forma eficaz se comparado a outras abordagens e resulta em arquiteturas que buscam a modularidade mais eficientemente.

## 2.3 Análise e Projeto

A análise e projeto de sistemas é um campo ativo e empolgante, no qual os analistas aprendem continuamente novas técnicas e abordagens para desenvolver sistemas de forma mais efetiva e eficiente. Os projetos exigem que os analistas reúnam os requisitos, modelem as necessidades da empresa e criem planos de como o sistema deve ser desenvolvido. Isto requer também um conhecimento de conceitos de comportamento organizacional, como o gerenciamento de mudanças e o desenvolvimento de equipes, aplicado nos projetos dos mais diversos segmentos e durante as fases pelas quais os projetos de *software* geralmente passam: planejamento, análise, projeto, implementação, testes e implantação. Essas fases compõem o ciclo de vida básico de desenvolvimento de sistemas, independente da abordagem ou metodologia utilizada (DENNIS, 2005; BEZERRA, 2002).

O ciclo básico de desenvolvimento é um processo gradual, no qual os resultados obtidos na fase de análise servem como entrada na fase de projeto. Por sua vez, a fase de implementação utiliza os resultados obtidos no projeto para produzir o sistema real. Esse ciclo de vida básico de desenvolvimento de sistemas é implementado nas várias metodologias de desenvolvimento de sistemas (DENNIS, 2005; BEZERRA, 2002).

O processo de desenvolvimento IBM *Rational Unified Process* (RUP) é um processo que está fortemente centrado na arquitetura, funcionalidade e desenvolvimento iterativo e incremental. O RUP se preocupa em retratar a arquitetura do sistema, ressaltando as características mais importantes do projeto. No entanto, o RUP é um processo configurável, o que permite que a organização que empregar o RUP poderá adequá-lo as suas necessidades.

O RUP, segundo Tonsig (2003), está baseado em um conjunto de princípios que são considerados fundamentais para o desenvolvimento de um *software*, onde, ainda conforme Tonsig (2003), as funcionalidades são demonstradas através de casos de uso, que são a representação dos requisitos funcionais.

O primeiro princípio diz respeito ao desenvolvimento iterativo do *software*, o qual facilita os ajustes táticos dos requisitos e de novas características. Além disso, é possível identificar antecipadamente os riscos do projeto, assim como algumas inconsistências no projeto, entre os requisitos e também na implementação. Com os riscos identificados, algumas alterações podem ser realizadas sem provocar maiores danos, o que facilita também a administração dos requisitos, que ainda assim podem sofrer modificações durante a elaboração do projeto de sistema.

A utilização de uma arquitetura baseada em componentes busca a reutilização e adaptação de componentes existentes em várias fontes. Esses componentes são módulos definidos na arquitetura e que possuem uma funcionalidade claramente delimitada.

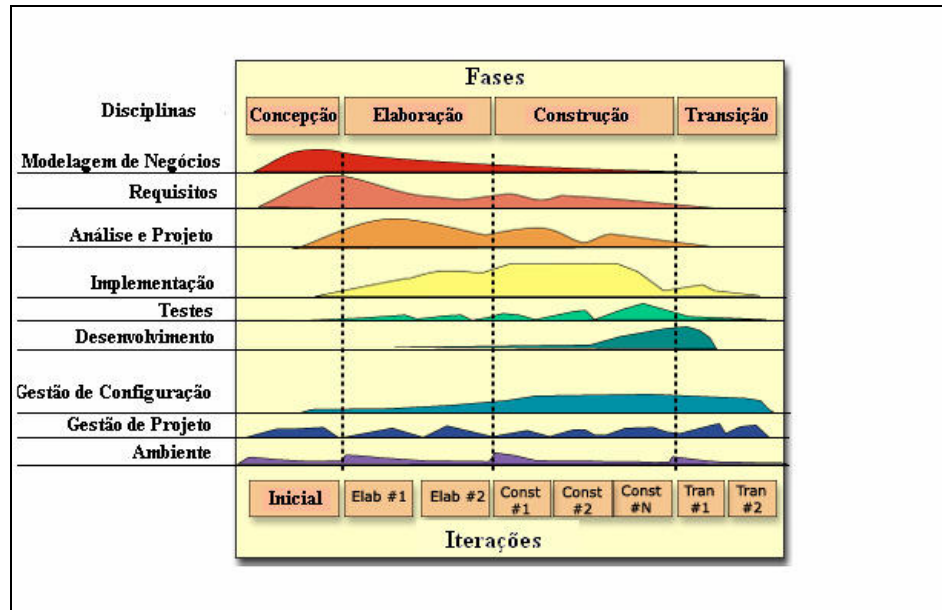
A modelagem visual do *software* consiste em criar modelos para auxiliar a equipe de desenvolvimento a visualizar, construir e documentar a estrutura e comportamento da arquitetura de um sistema. Utilizar uma linguagem de modelagem padrão permite que membros da equipe se comuniquem de forma clara e sem ambigüidades.

A qualidade de *software* é fundamental na construção e manutenção do *software* e, portanto, deve ser contínua em relação à funcionalidade, confiabilidade e validação.

Por último, temos a gestão das alterações no *software*, na qual é realizada a coordenação das atividades que originam os módulos e suas respectivas versões.

Assim, a metodologia RUP contempla as atividades que vão desde o planejamento do projeto até os processos de teste. Essa contemplação determina o quanto o método é adequado à construção de sistemas de maneira a atender os parâmetros de qualidade e

produtividade. Além disso, o processo de desenvolvimento que utiliza o RUP se dá mediante uma série de ciclos que constituem uma versão do produto de *software*, onde cada ciclo é composto de quatro fases: concepção, elaboração, construção e transição. Em cada uma dessas fases é que se realizam as iterações, que abrangem um série de atividades, como mostra a Figura 5 (TONSIG, 2003):



**Figura 5. Fases da Metodologia RUP**

Fonte: Adaptado da Rational (2006).

As fases que contemplam a estrutura geral do processo de desenvolvimento são descritas a seguir:

- **Concepção:** inicialmente, é preciso estabelecer o escopo do projeto e sua viabilidade econômica.
- **Elaboração:** o objetivo da fase de elaboração é buscar identificar e eliminar os riscos, estabelecer uma arquitetura de fundação sólida a partir da qual o sistema a ser projetado poderá evoluir.
- **Construção:** durante a fase de construção, conforme o próprio nome indica, ocorre o desenvolvimento do produto, que é feito de forma iterativa e incremental até sua conclusão.

- Transição: depois da construção, o produto é colocado em uso, o que naturalmente leva a surgir novas considerações que irão demandar a construção de novas versões. Essas versões, por sua vez, podem acontecer por ajustes do sistema, correção de problemas ou conclusão de algumas características que foram postergadas.

É importante lembrar que dentro de cada fase um conjunto de iterações (planejamento, levantamento de requisitos, análise e projeto, implementação e testes) é realizado. Portanto, na fase de concepção o foco está no planejamento e levantamento dos requisitos, ou seja, no entendimento dos requisitos e na determinação de um escopo para o projeto. Na fase de elaboração é realizada a modelagem dos requisitos, isto é, o levantamento e análise, bem como iniciam-se também alguns trabalhos nas atividades de análise do projeto e também na implementação, como a prototipação da arquitetura. Na fase de construção o enfoque ficará concentrado na análise e na implementação, visando evoluir o protótipo inicial de arquitetura até obter o primeiro produto operacional. Por fim, na fase de transição, os testes são o destaque, pois é preciso garantir que o sistema possua o nível adequado de qualidade. Nesta fase também acontece o treinamento dos usuários, durante o qual eles podem auxiliar com argumentos em pontos de melhoria do sistema ou ainda ajustar características do software (TONSIG, 2003).

## **2.4 Unified Modeling Language**

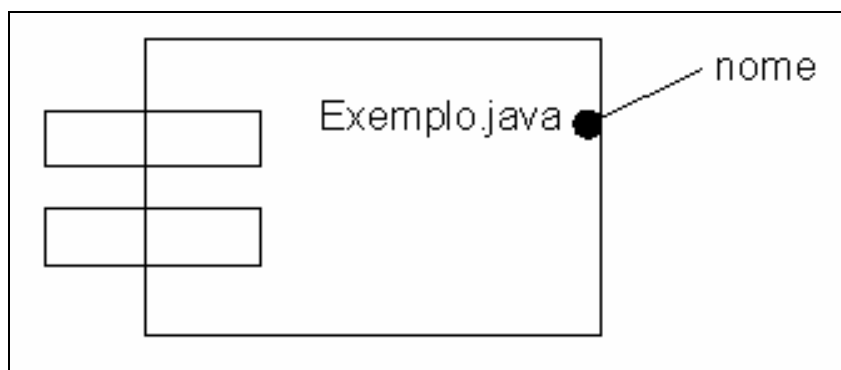
A *Unified Modeling Language* (UML) é uma linguagem-padrão para a elaboração da estrutura de projetos de *software*. Pode ser utilizada para visualização, especificação,

construção e documentação de elementos pertencentes a sistemas complexos de *software*. É uma linguagem muito expressiva, pois abrange todas as visões necessárias para o desenvolvimento e implantação de sistemas (BOOCH, 2000).

A linguagem de modelagem UML é somente uma parte do método para o desenvolvimento de *software*, sendo independente do processo de desenvolvimento utilizado, apesar de se comportar muito bem com processos orientados a casos de uso centrado na arquitetura, iterativo e comportamental. Além disso, a UML é uma linguagem para visualização, especificação, construção e documentação de todo tipo de sistemas (BOOCH, 2000).

Os diagramas e notações apresentados pela UML descrevem todos os elementos de um sistema. Para o desenvolvimento da monografia serão utilizados muitos desses diagramas e notações, mas principalmente aqueles que representam componentes na UML.

Para Booch (2000, p.20) “os componentes são partes físicas e substituíveis de um sistema, que proporcionam a realização de um conjunto de interfaces”. Tipicamente os componentes representam o pacote físico de elementos lógicos diferentes, como classes, interfaces e colaborações. Graficamente, os componentes são representados como retângulos com abas, incluindo apenas seus nomes, como mostra a Figura 6:



**Figura 6. Representação de Componente em UML**

Fonte: Adaptado de BOOCH, 2000.

Além disso, essas características são enfatizadas por Booch (2000, p. 347):

“Primeiro, um componente é *físico*. Ele vive no mundo dos bits e não dos conceitos. Segundo, um componente é *substituível*. Um componente é substituível – é possível substituir um componente por outro que esteja em conformidade com as interfaces. Terceiro, um componente é *parte de um sistema*. Um componente raramente existe sozinho. Em vez disso, um determinado componente colabora com outros componentes e, ao fazer isso, existe no contexto de arquitetura ou da tecnologia em que se pretende utilizá-lo. Um componente é lógica e fisicamente coeso e, portanto, denota uma parte estrutural e/ou comportamental significativa de um sistema maior. Um componente poderá ser reutilizado em muitos sistemas”.

Um componente se adapta ao sistema e expõem um conjunto de interfaces. As interfaces, portanto, constituem a ponte entre os modelos lógico e físico. Os componentes são semelhantes às classes, ambos tem nomes, ambos podem expor um conjunto de interfaces, ambos podem participar de um relacionamento de dependência, generalização e associação. Mas existem diferenças significativas, conforme Booch (2000):

**Quadro 2. Diferenças entre Classe e Componente**

<b>Classes</b>	<b>Componentes</b>
Representam abstrações lógicas;	Representam coisas físicas que vivem no mundo dos bits;
Podem ter atributos e operações diretamente;	Geralmente componentes somente têm operações que são alcançados por meio das suas interfaces;
	Representam o pacote físico de componentes e se apresentam em um nível diferente de abstração;

Para especificar um serviço de um componente ou de uma classe, utilizamos a interface. Uma interface é uma coleção de operações utilizadas por uma classe ou um componente realizar os acessos a outros componentes e/ou classes (BOOCH, 2000).

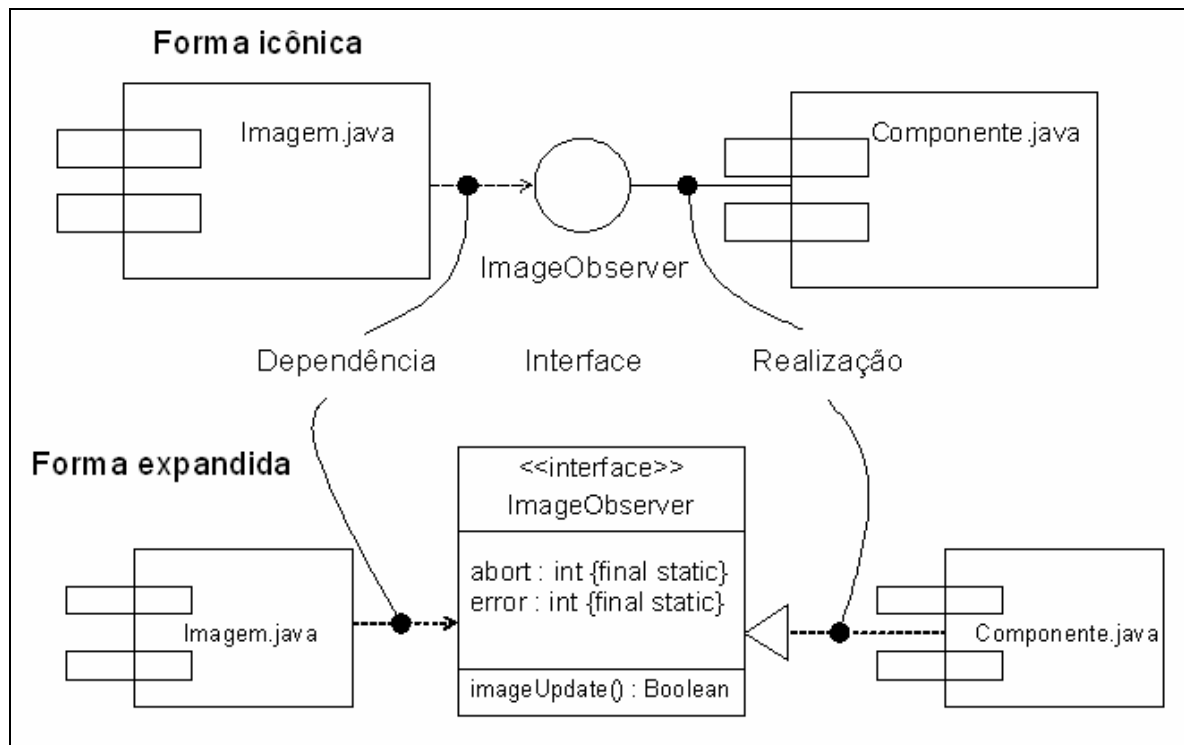
O relacionamento de um componente com suas interfaces é feito pela representação da interface em sua forma icônica, oculta e da interface em sua forma expandida, em alguns casos revelando suas operações. Nos dois casos, o componente tem acesso aos serviços do outro componente por meio da interface. A conexão entre o componente e sua interface é realizada usando um relacionamento de dependência (BOOCH, 2000, p. 346). A Figura 7 mostra essa conexão.

Booch (2000, p. 346) explica também como são chamadas as interfaces segundo o serviço que realizam:



“Uma interface realizada por um componente é chamada *interface de exportação*, significando uma interface em que o componente fornece um serviço para outros componentes. Um componente poderá fornecer muitas interfaces de exportação. A interface utilizada pelo componente é chamada de *interface de importação*, significando uma interface à qual o componente se adapta e a partir da qual é construído. Um componente poderá estar em conformidade a muitas interfaces de importação. Além disso, um componente pode tanto importar, como exportar interfaces.

Uma determinada interface poderá ser exportada por um componente e importada por um outro. O fato dessa interface se encontrar entre dois componentes quebra a dependência direta entre os componentes. Um componente que utiliza uma determinada interface funcionará adequadamente, qualquer que seja o componente que a realiza. É claro que um componente pode ser utilizado em um contexto, se e somente se todas as interfaces de importação forem fornecidas pelas interfaces de exportação de outros componentes”.



**Figura 7. Componentes e Interfaces**

Fonte: Adaptado de BOOCH (2000).

A importância e facilidade do uso de componentes é confirmada por Booch (2000, p. 346-347):

“O propósito básico de qualquer facilidade de um sistema operacional baseado em componentes consiste em permitir a montagem de sistemas a partir de partes binárias substituíveis. Isso significa que é possível criar um sistema a partir de componentes e depois evoluir esse sistema pela adição de novos componentes e pela substituição dos anteriores, sem reconstruir o sistema. As interfaces são recurso-chave para que isso possa acontecer. Ao especificar uma interface, você pode incluir no sistema executável qualquer componente que esteja com conformidade ou que forneça essa interface. Você pode estender o sistema, fazendo com que os componentes ofereçam novos serviços por meio de outras interfaces, as quais, por sua vez, podem ser descobertas e utilizadas por outros componentes”.

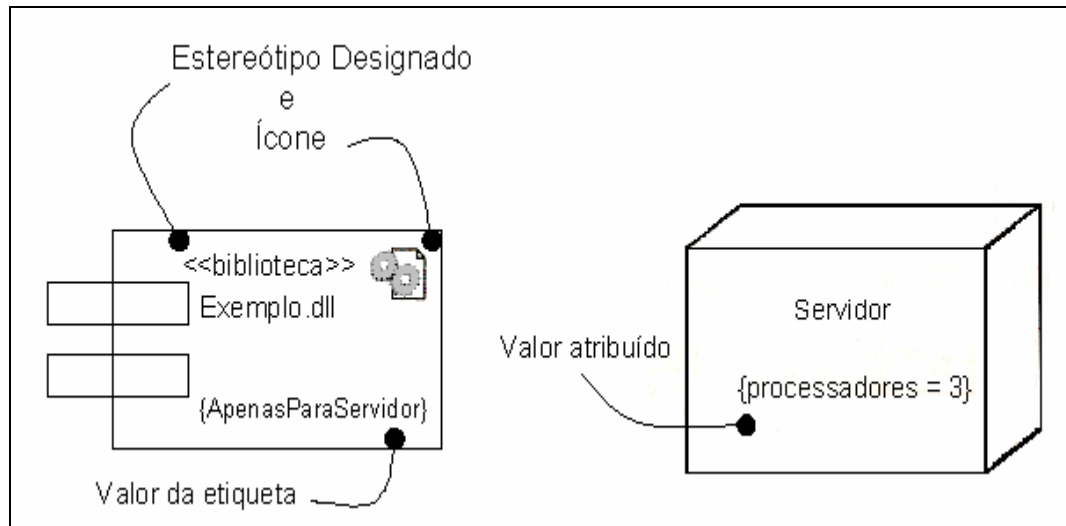
Existem três tipos de componentes, segundo Booch (2000):

- Componentes de Implantação: Necessários para formar sistemas executáveis, como bibliotecas dinâmicas (DLLs) e os executáveis (EXEs).
- Componentes do Produto do Trabalho: São essencialmente o resíduo do processo de desenvolvimento, formado por arquivos de código-fonte e arquivos de dados. Utilizados para a criação dos componentes de implantação, e por sua vez, do sistema executável.
- Componentes de Execução: São objetos instanciados a partir de DLLs, ou seja, criados por consequência de um sistema em execução.

Todos os mecanismos de extensibilidade da UML se aplicam aos componentes. Entretanto, para isso, é necessário o uso de valores atribuídos para estender as propriedades dos componentes e estereótipos para especificar novos tipos de componentes (BOOCH, 2000).

Um estereótipo é representado como um nome entre ângulos (<<nome>>), colocado acima do nome de outro elemento. Como uma indicação visual, você pode definir um ícone para estereótipo e apresentá-lo à direita do nome.

Já o valor atribuído é representado como uma sequência de caracteres entre chaves. Essa sequência inclui um nome (a etiqueta), um separador (o símbolo =) e um valor atribuído. A Figura 8 demonstra como o estereótipo e o valor atribuído é representado:



**Figura 8. Estereótipos e Valores atribuídos**  
 Fonte: Adaptado de BOOCH (2000).

Além disso, a UML define cinco estereótipos-padrão que se aplicam aos componentes:

- Executável: especifica um componente que pode ser executado em um nó.
- Biblioteca: especifica uma biblioteca de objetos estática ou dinâmica.
- Tabela: especifica um componente que representa uma tabela de banco de dados.
- Arquivo: especifica um componente que representa um documento contendo código-fonte ou dados.
- Documento: especifica um componente que representa um documento.

Contudo, além do diagrama de componentes, os diagramas de classes, os diagramas de casos de uso, os diagramas de sequência e os diagramas de atividade também são usados para o desenvolvimento da modelagem do sistema e do componente.

## 2.5 Frameworks

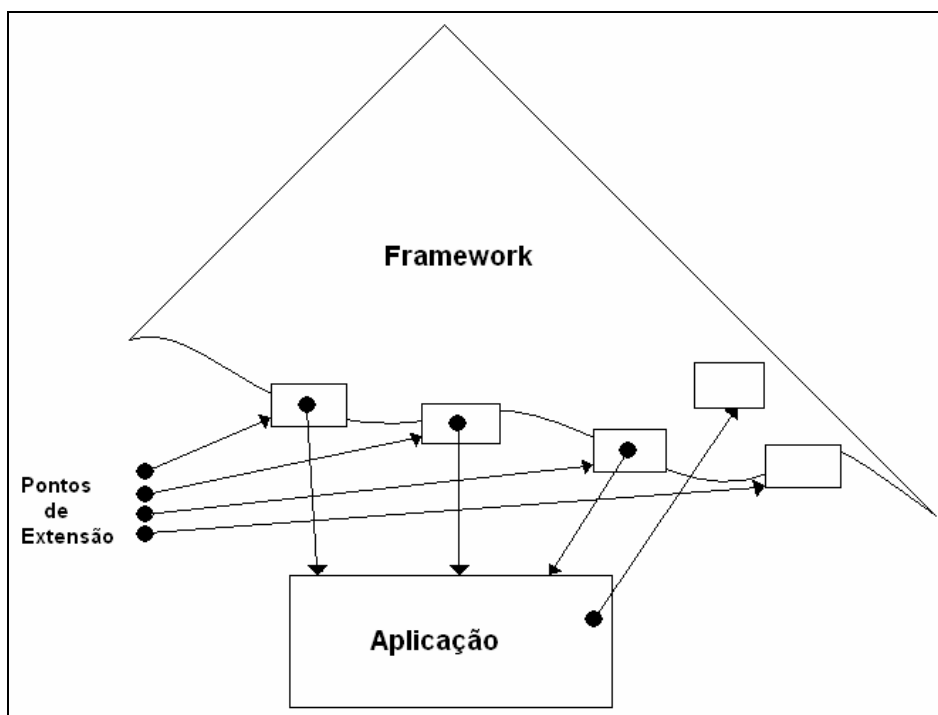
Um *framework*, para Peters (2001), é um sistema modificável para propósitos gerais, que ajuda a estreitar a distância entre uma resolução de alto nível de um problema e a sua implementação em *software*. Fornece ainda uma forma conveniente para a estruturação, combinação de dados e estruturas de controle.

Segundo Sommerville (2003), o *framework* é um projeto constituído de um conjunto de classes e da interface entre elas. Detalhes específicos do projeto são implementados com o acréscimo de componentes e o fornecimento da implementação concreta das classes abstratas nos *frameworks*. Os *frameworks* raramente são aplicações propriamente ditas. As aplicações normalmente são construídas pela integração de diversos *frameworks*.

Para Sommerville (2003), os *frameworks* dividem-se em três classes:

- *Frameworks* de infra-estrutura de sistema: São compatíveis com o desenvolvimento das infra-estruturas de sistemas, como comunicações, interfaces com o usuário e compiladores.
- *Frameworks* de integração com *middleware*: Consiste em um conjunto de classes de objetos-padrão e associadas, que aceitam a comunicação de componentes e a troca de informações.
- *Frameworks* de aplicações corporativos: Ocupam-se de domínios específicos de aplicações, como telecomunicações ou sistemas financeiros. Incluem o conhecimento de domínio e são compatíveis com o desenvolvimento de aplicações para o usuário final. Normalmente mais abstratos, possibilitando a criação de uma gama mais ampla de aplicações.

“Um *framework* é formado por uma estrutura de classes contendo pontos de extensão. Estes pontos podem ser estendidos pela aplicação, conforme pode ser visto na figura a seguir. Desta forma um determinado programa pode ser visto como sendo formado por um *framework* provendo serviços genéricos e que são devidamente especializados por elementos da aplicação. Contrário ao caso de componentes em que a aplicação comanda e coordena o funcionamento dos componentes, um *framework* comanda e coordena o comportamento da aplicação” (STAA, 2000, p. 330).



**Figura 9. Interação entre um *Framework* e a *Aplicação***

Fonte: STAA, 2000, p. 331.

Para Booch (2000), um *framework* é um padrão de arquitetura que fornece um *template* extensível para aplicações dentro de um domínio. Ao especificar um *framework*, você especifica o esqueleto de uma arquitetura, juntamente com os conectores, guias, botões e indicadores que são expostos aos usuários que desejam adaptar esse *framework* ao seu próprio contexto.

Na mesma idéia, Larman (2000) define *framework* como um conjunto coeso de classes que colaboram para o núcleo invariante de um subsistema lógico. Essas classes são concretas e abstratas, sendo que as últimas definem interfaces a serem seguidas, interações entre objetos e outros invariantes. Além disso, *frameworks* fornecem um grau muito elevado de reutilização, muito mais do que classes individuais.

Dessa forma, usar *frameworks* prontos ou criar os próprios são caminhos que as organizações podem seguir para reutilizar código e reduzir o tempo de implementação e entrega de sistemas. No entanto, existem outros caminhos, como mencionado anteriormente, que utilizam conceitos de modularização e componentização nos processos de desenvolvimento de sistemas.

## **2.6 Considerações Finais**

A revisão bibliográfica permitiu entender conceitos que são fundamentais para o desenvolvimento da modelagem, análise e implementação do sistema e do componente.

Os autores citados durante toda a revisão foram fundamentais para comprovar a importância da modularização e componentização no desenvolvimento de sistemas que permitem o reuso e a redução de custos no desenvolvimento e manutenção do sistema.

A metodologia de desenvolvimento estudada foi adotada pela abordagem que o RUP tem em relação a componentes e também pelas características de customização, sendo adaptável a cada projeto.

Enfim, todos os critérios abordados nesse estudo são colocados em prática a partir do próximo capítulo, onde é descrito o desenvolvimento da modelagem e etapas posteriores, para então, chegar ao produto final.

### 3 MODELAGEM

A modelagem de um sistema de *software*, assim como a modelagem de componentes possui uma característica intrínseca: a complexidade de seu desenvolvimento. Para isso faz-se uso de alguns recursos que auxiliam no planejamento, construção e implantação desses sistemas (BEZERRA, 2002).

A proposta de solução utiliza uma customização do processo de desenvolvimento de *software* IBM *Rational Unified Process* (RUP) e é representada através da Linguagem de Modelagem Unificada (UML).

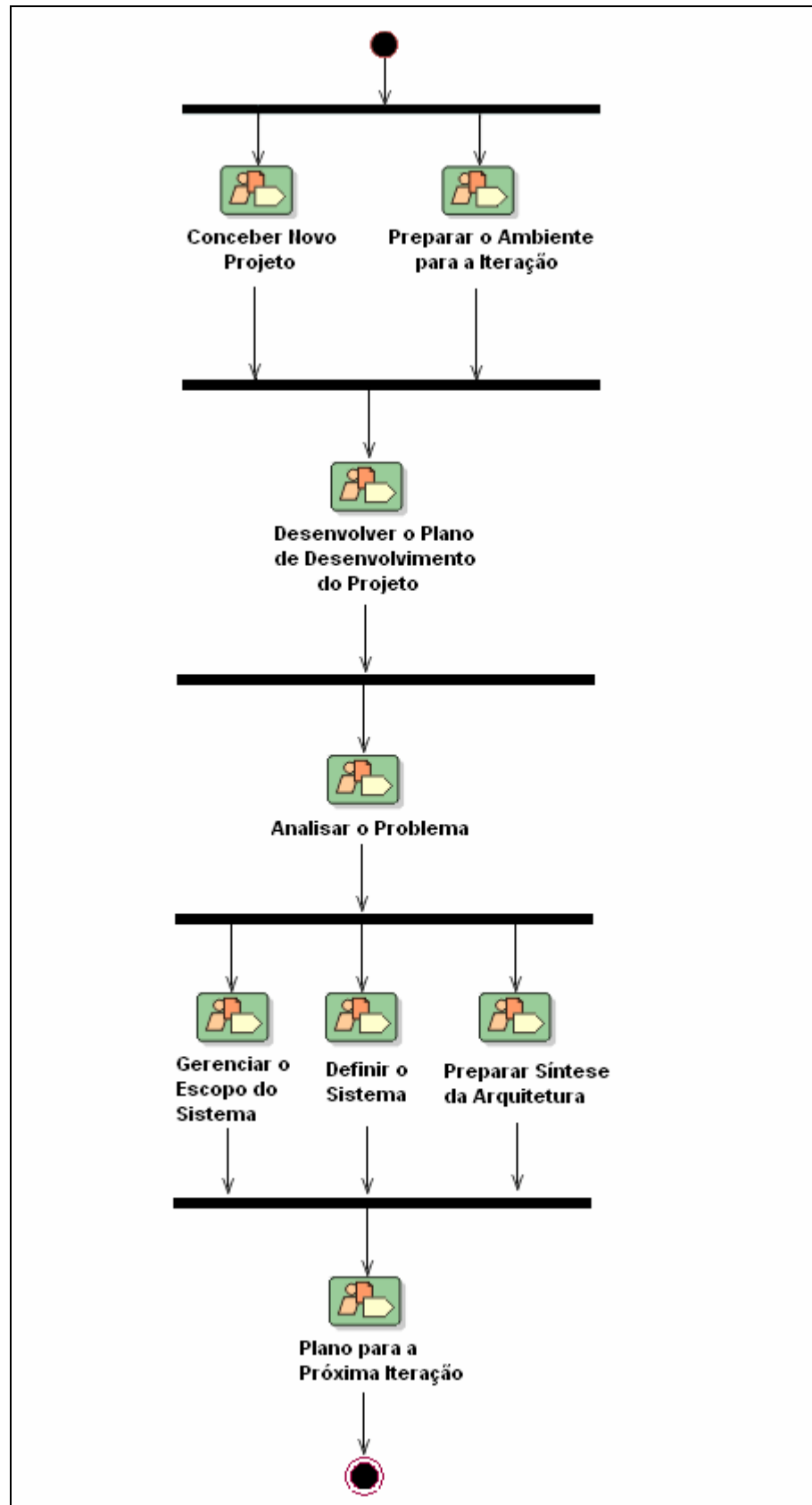
Para o processo de análise e projeto do sistema foi feito uso da ferramenta de modelagem *Enterprise Architect* (EA). Segundo a Rational (2006), nesta fase o foco está em compreender os elementos-chave do processo, com o objetivo de capturar todas as informações necessárias para um desenvolvimento coerente com a realidade do problema.

Neste capítulo serão descritas as fases da metodologia RUP, assim como suas atividade e artefatos. Estes últimos, serão detalhados nos apêndices deste trabalho. Por fim, o desenvolvimento se dará em uma visão mais abstrata, chegando a níveis de detalhamento bem específico e singular.

### **3.1 Concepção**

A fase de concepção aborda uma visão do sistema e estabelece um projeto formal para construí-lo. Além disso, serão definidos os principais casos de uso, que devem ser elaborados com precisão, para então proceder com as estimativas de prazo e custo. Assim, a ênfase nessa etapa recai sobre o planejamento e por conseguinte, é necessário levantar os requisitos do sistema e analisá-los.





**Figura 10. Workflow Fase de Concepção**  
 Fonte: Adaptado da Rational (2006).

As tarefas executadas na fase de Concepção foram:

### **Conceber Novo Projeto**

- Iniciar Projeto

A equipe planeja o projeto e define os critérios para o sucesso de medição do projeto (RATIONAL, 2006).

Responsável: Gerente de Projeto.

### **Preparar o Ambiente para a Iteração**

- Verificar Configuração e Instalação de Ferramentas

Selecionar e adquirir as ferramentas necessárias para a realização do projeto (RATIONAL, 2006).

Responsável: Especialista em Ferramentas.

### **Desenvolver o Plano de Desenvolvimento do Projeto**

- Definir Organização do Projeto e a Equipe

Definir uma estrutura organizacional para o projeto, a equipe e suas exigências conforme as estimativas do esforço - em termos de números, tipos e níveis da experiência - para a próxima iteração e para as iterações subsequentes (RATIONAL, 2006).

Responsável: Gerente de Projeto.

- Planejar Fases e Iterações

Estimar o espaço, o esforço e o custo totais para o projeto. Definir um conjunto de iterações dentro das fases do projeto, e identificar os objetivos para cada uma destas iterações. Desenvolver o cronograma e o orçamento para o projeto, verificando as atividades iniciais para a conclusão do projeto (RATIONAL, 2006).

Responsável: Gerente de Projeto.

- Revisar Planejamento de Projeto

Aprovar o Plano inicial de Desenvolvimento do *Software*. Se houver mudanças é necessário realizar uma nova revisão no Plano e em seguida aprová-las (RATIONAL, 2006).

Responsável: Gerente de Projeto.

### **Analisar o Problema**

- Encontrar Atores e Casos de Uso

Definir o escopo e a funcionalidade do sistema - o que será feito pelo sistema e o que não será feito pelo sistema, definindo quem e o que interagirá com o sistema. Criar diagramas de casos de uso (RATIONAL, 2006).

Responsável: Analista de Sistema.

- Capturar um Vocabulário Comum

Definir um vocabulário comum que possa ser usado em todas as descrições textuais do sistema, especialmente em descrições dos casos de uso (RATIONAL, 2006).

Responsável: Analista de Sistema.

### **Gerenciar o Escopo do Sistema**

- Priorizar Casos de Uso

Definir a entrada para seleção dos cenários e casos de uso que devem ser analisadas na iteração atual. Definir os cenários e casos de uso que representam alguma funcionalidade significativa. Definir os cenários e casos de uso que têm uma cobertura arquitetural substancial ou que forçam ou ilustram um ponto específico delicado da arquitetura (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Gerenciar Dependências

Usar atributos e habilidades de exigências do projeto, para ajudar no gerenciamento do escopo e de mudanças no projeto (RATIONAL, 2006).

Responsável: Analista de Sistema.

### **Definir o Sistema**

- Encontrar Atores e Casos de Uso

Definir o escopo e a funcionalidade do sistema - o que será feito pelo sistema e o que não será feito pelo sistema, definindo quem e o que interagirá com o sistema. Criar diagramas de casos de uso (RATIONAL, 2006).

Responsável: Analista de Sistema.

- Capturar Vocabulário Comum

Descrição da atividade: Definir um vocabulário comum que possa ser usado em todas as descrições textuais do sistema, especialmente em descrições dos casos de uso (RATIONAL, 2006).

Responsável: Analista de Sistema.

- Gerenciar Dependências

Usar atributos e habilidades de exigências do projeto, para ajudar no gerenciamento do escopo e de mudanças no projeto (RATIONAL, 2006).

Responsável: Analista de Sistema.

### **Preparar Síntese da Arquitetura**

- Analisar a Arquitetura

Definir uma arquitetura candidata para o sistema baseado na experiência adquirida dos sistemas similares ou em domínios similares do problema. Definir testes padrões da arquitetura, os mecanismos chaves, e modelar convenções para o sistema (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Analisar os Casos de Uso

Identificar as classes que executam o fluxo dos eventos dos casos de uso. Distribuir o comportamento dos casos de uso para as classes, usando realizações de casos de uso. Identificar as responsabilidades, os atributos e as associações das classes. Anotar o uso de mecanismos arquiteturais (RATIONAL, 2006).

Responsável: Projetista.

### **Plano para a Próxima Iteração**

- Desenvolver Plano de Iteração

Criar e aprovar as metas propostas para a iteração da próxima fase (RATIONAL, 2006).

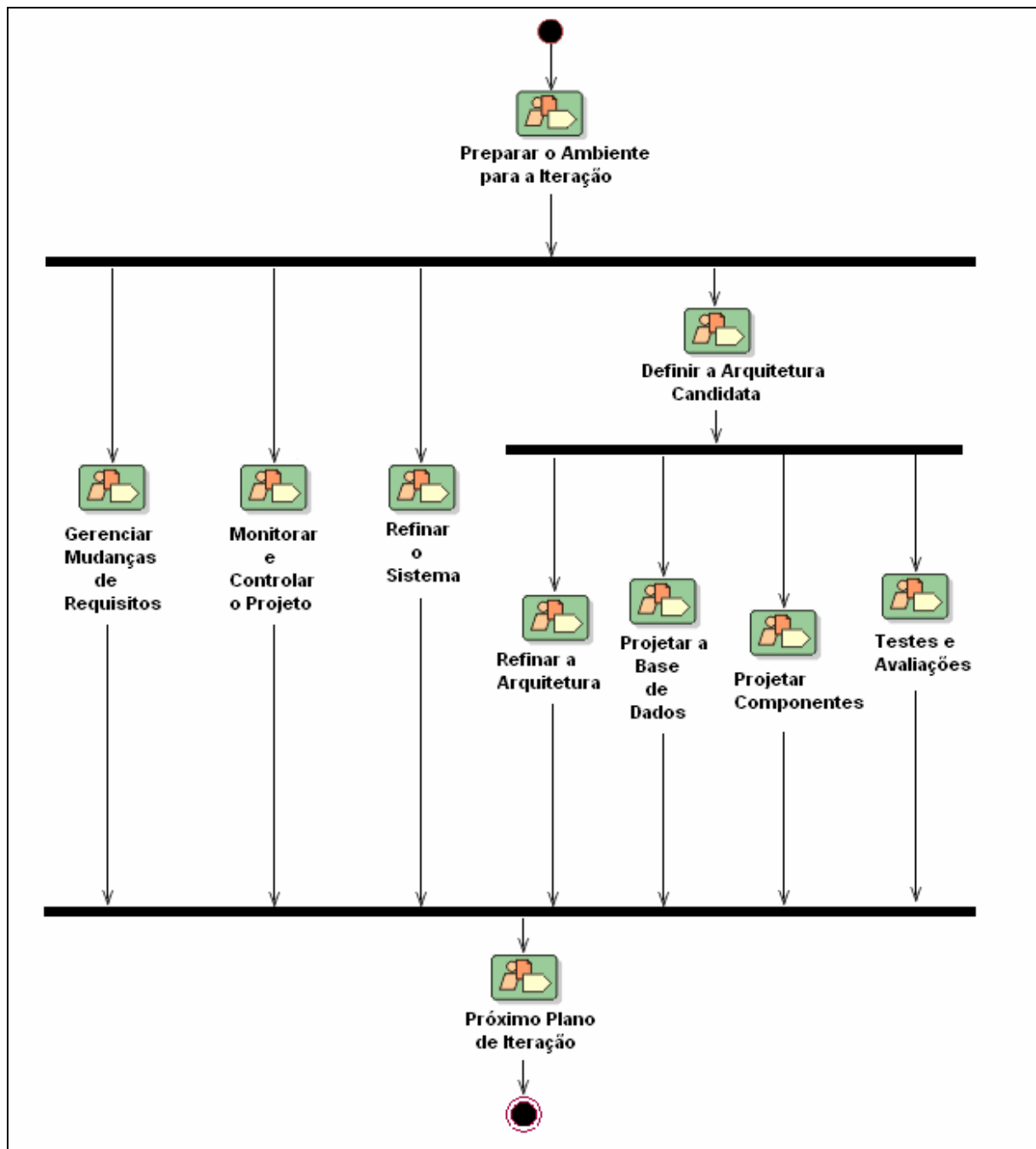
Responsável: Gerente de Revisão.

Os artefatos gerados durante a fase de Concepção foram: Plano de Desenvolvimento de *Software* (APÊNDICE A), Plano de Iteração da Concepção (APÊNDICE F), Plano de Iteração Elaboração (APÊNDICE F), Glossário (APÊNDICE J), Especificação Complementar (APÊNDICE B), Documento de Arquitetura do *Software* (APÊNDICE E), Especificação dos Casos de Uso (APÊNDICE D), Especificação da Realização dos Casos de Uso (APÊNDICE G).

## **3.2 Elaboração**

O objetivo da fase de Elaboração é definir a arquitetura candidata de um sistema para prover uma base estável para a carga de projeto e esforço de implementação que será

realizado na fase de Construção. A arquitetura evolui a partir da consideração dos requisitos mais significativos (dos que possuem um grande impacto na arquitetura de um sistema) e um cálculo do risco. A estabilidade da arquitetura será avaliada através de um ou mais protótipos arquiteturais (RATIONAL, 2006).



**Figura 11. Workflow Fase de Elaboração**

Fonte: Adaptado da Rational (2006).

As tarefas executadas na fase de Elaboração foram:

**Preparar o Ambiente para a Iteração**

- Verificar Configuração e Instalação de Ferramentas

Selecionar e adquirir as ferramentas necessárias para a realização do projeto (RATIONAL, 2006).

Responsável: Especialista em Ferramentas.

### **Gerenciar Mudanças de Requisitos**

- Estruturar os modelos de caso de uso

Extrair os comportamentos através de casos de uso abstratos, caso isso seja necessário e encontrar novos atores abstratos que definem os papéis que são compartilhados por diversos atores (RATIONAL, 2006).

Responsável: Analista de Sistemas.

- Gerenciar as dependências

Usar atributos e características dos requisitos do projeto para auxiliar no gerenciamento do escopo e nas mudanças de requisitos que envolvem o projeto (RATIONAL, 2006).

Responsável: Analista de Sistemas.

- Revisar os requisitos

Verificar formalmente se os requisitos estão de acordo com as idéias do cliente (RATIONAL, 2006).

Responsável: Revisor Técnico.

### **Monitorar e Controlar o Projeto**

- Monitorar a situação do projeto

Captura o status atual do projeto e avalia a situação atual do cronograma (RATIONAL, 2006).

Responsável: Gerente de Projeto.

- Planejar e designar o trabalho

Aprovar as mudanças (defeitos) que são levantadas durante a iteração e adaptá-las ao produto e/ou processo (RATIONAL, 2006).

Responsável: Gerente de Projeto.

- Tratar os problemas e exceções

Propor ações corretivas apropriadas para resolver os problemas e as exceções que são levantados no projeto (RATIONAL, 2006).

Responsável: Gerente de Projeto.

### **Refinar o Sistema**

- Detalhar o Caso de Uso

Descrever um ou mais casos de uso, detalhando o suficiente para o desenvolvimento do *software* (RATIONAL, 2006).

Responsável: Especificador de Requisitos.

- Detalhar os Requisitos do Sistema

Coletar, detalhar e organizar o pacote dos artefatos que descrevem completamente as exigências do sistema ou do subsistema (RATIONAL, 2006).

Responsável: Especificador de Requisitos.

### **Definir a Arquitetura candidata**

- Analisar a Arquitetura

Descrição da atividade: Definir uma arquitetura para o sistema baseado na experiência ganha com sistemas similares ou em domínios similares do problema. Definir testes, o padrão de arquitetura, os mecanismos chaves, e modelar convenções para o sistema (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Analisar os Casos de Uso



Identificar as classes que encadeiam os eventos do fluxo de casos de uso. Distribuir o comportamento dos casos de uso para aquelas classes, usando as realizações dos casos de uso. Identificar as responsabilidades, os atributos e as associações das classes. Anotar o uso de mecanismos arquiteturais (RATIONAL, 2006).

Responsável: Projetista.

### **Refinar a Arquitetura**

- Identificar os Mecanismos do Projeto

Refinar os mecanismos da análise do projeto baseado nas restrições impostas pelo ambiente da execução (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Identificar os Elementos do Projeto

Analisar as interações classes de análise para identificar os elementos do projeto (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Incorporar Elementos existentes no Projeto

Analisar as interações das classes de análise para encontrar interfaces, classes do projeto e subsistemas do projeto. Refinar a arquitetura, incorporando o reuso quando possível. Identificar soluções comuns aos problemas do projeto (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Estruturar o Modelo de Implementação

Estabelecer uma estrutura na qual a implementação residirá. Atribuir responsabilidades para implementação de subsistemas e seus conteúdos (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Descrever a Arquitetura em Tempo de Funcionamento

Analisar requisitos concorrentes, identificar processos, identificar mecanismos de comunicação *inter-process*, alocar recursos de coordenação *inter-process*, identificar ciclo de vida dos processos e distribuir elementos modelo entre os processos (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

### **Projetar a Base de Dados**

- Projetar Base de Dados

Assegurar que os dados persistentes sejam armazenados consistentemente e eficientemente. Definir o comportamento que deve ser implementado na base de dados (RATIONAL, 2006).

Responsável: Projetista de Base de Dados.

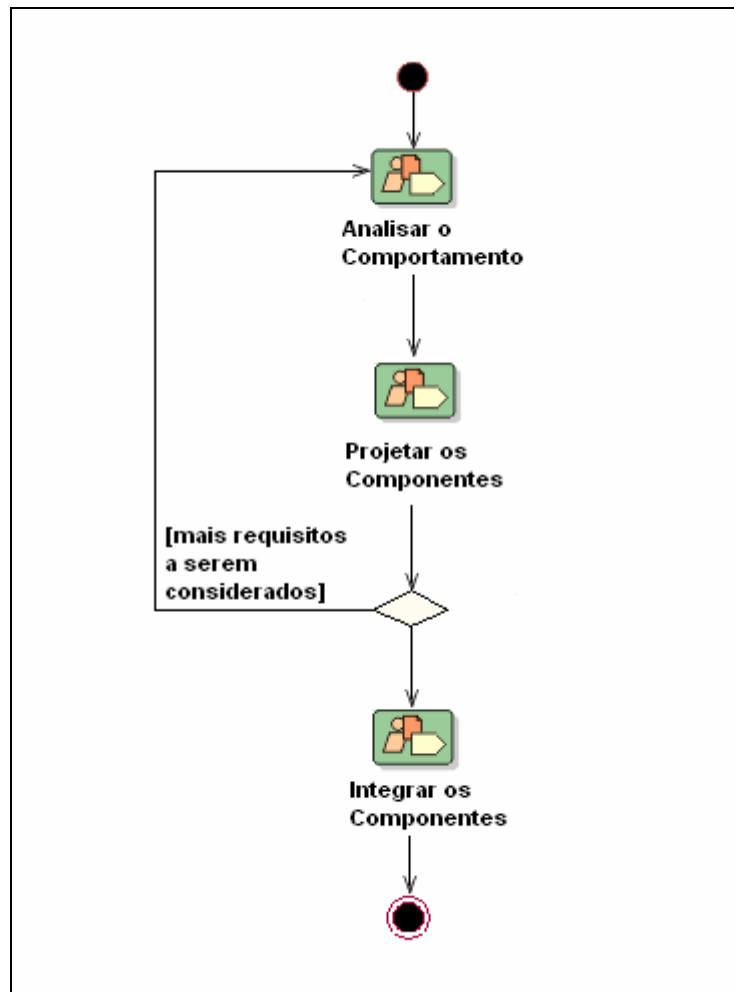
- Projetar Classes

Assegurar de que as classes forneçam o comportamento das realizações dos casos de uso. Assegurar que as informações fornecidas não exerçam ambigüidade à classe. Capturar os requisitos não-funcionais relacionados às classes. Incorporar os mecanismos do projeto usados pelas classes (RATIONAL, 2006).

Responsável: Projetista.

### **Projetar os Componentes**

A tarefa projetar componentes envolve uma série de outras tarefas que são necessárias para realizar o projeto de componentes de um sistema. Como a proposta de solução envolve o uso de um componente, este item é descrito na figura a seguir.



**Figura 12. Workflow da Tarefa Projetar Componentes**

Fonte: Adaptado da Rational (2006).

As atividades que envolvem o projeto de componentes são descritas a seguir conforme cada tarefa.

#### Analisar o Comportamento

- Analisar os Casos de Uso

Identificar as classes que executarão os fluxos de eventos dos casos de uso, distribuir o comportamento dos casos de uso para essas classes, usando a realização dos casos de uso. Identificar as responsabilidades, atributos e associações das classes (RATIONAL, 2006).

Responsável: Projetista.

- Identificar os Elementos do Projeto

Analisar as interações da classe de análise para identificar o elemento modelo do projeto (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Projetar as Interfaces com o Usuário

Produzir um projeto de interface com o usuário que apresente uma explicação sobre o funcionamento do sistema (RATIONAL, 2006).

Responsável: Projetista.

### Projetar os Componentes

- Projetar Casos de Uso

Refinar a realização dos casos de uso de acordo com as interações; refinar os requisitos nas operações do projeto das classes; refinar os requisitos nas operações do projeto de subsistemas e/ou suas interfaces; refinar os requisitos nas operações do projeto de módulos (RATIONAL, 2006).

Responsável: Projetista.

- Projetar Classes

Assegurar de que as classes forneçam o comportamento das realizações dos casos de uso. Assegurar que as informações fornecidas não exerçam ambigüidade a classe. Capturar os requisitos não-funcionais relacionados às classes. Incorporar os mecanismos do projeto usados pelas classes (RATIONAL, 2006).

Responsável: Projetista.

- Projetar Subsistemas

Definir os comportamentos especificados nas interfaces do subsistema de acordo com os elementos contidos no projeto e de subsistemas/interfaces externos; documentar a estrutura interna do subsistema; definir realizações entre as interfaces do subsistema e as classes contidas; determinar as dependências com outros subsistemas (RATIONAL, 2006).

Responsável: Projetista.

- Projetar Módulos

Definir e elaborar os tipos de módulos (RATIONAL, 2006).

Responsável: Projetista.

#### Integrar os Componentes

- Integrar os Subsistemas

Integrar os elementos em um subsistema, a seguir realizar a integração do subsistema ao sistema (RATIONAL, 2006).

Responsável: Projetista.

### **Testes e Avaliações**

- Executar os Testes

Criar e executar um ou mais artefatos de teste, que permitam a validação do produto de *software* com a execução física. Desenvolver os testes que podem ser executados conjuntamente com outros testes como parte de uma infraestrutura maior de teste (RATIONAL, 2006).

Responsável: Verificador.

### **Próximo Plano de Iteração**

- Desenvolver o Plano de Iteração

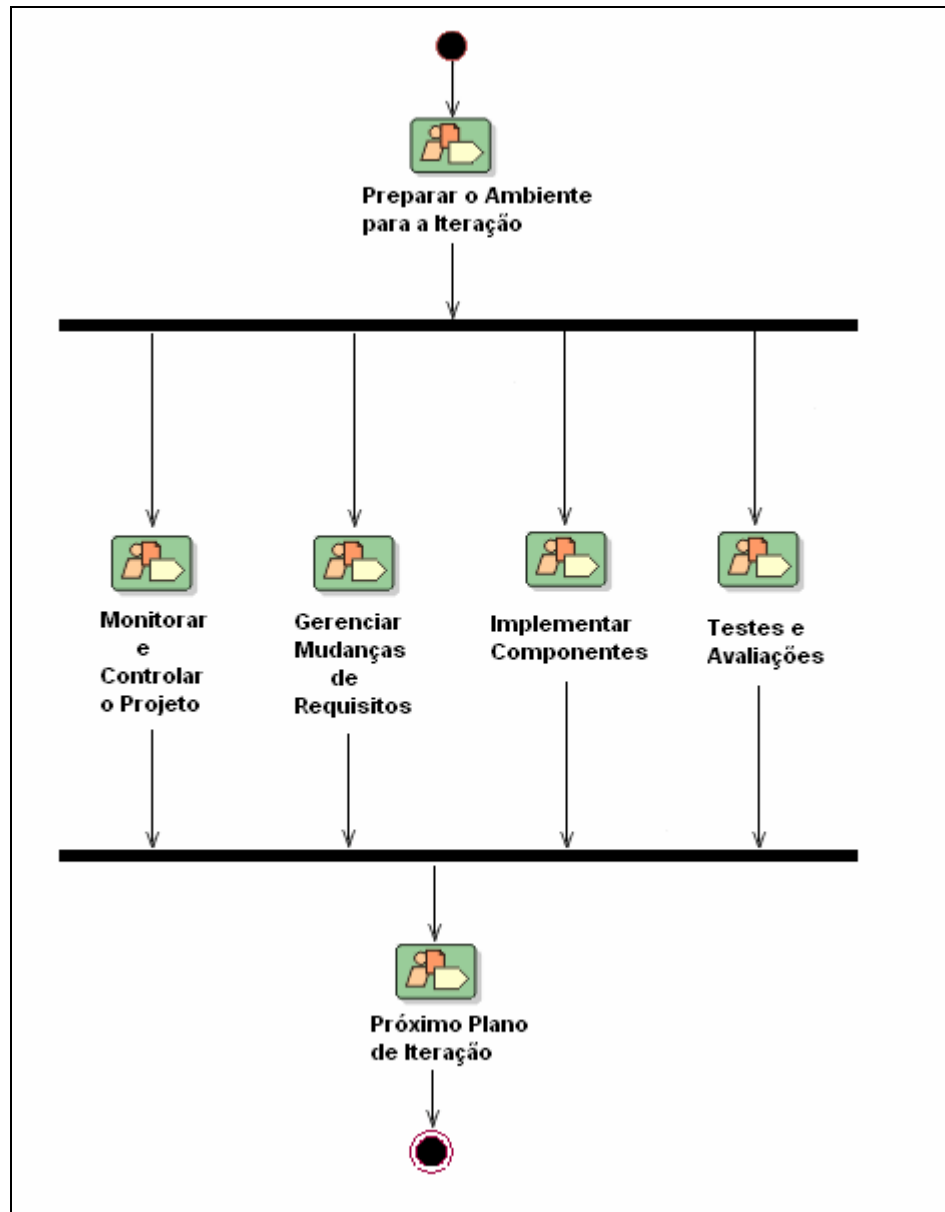
Desenvolver um refinado plano que busca detalhar os pacotes de trabalho e seus respectivos responsáveis, as datas previstas para a entrega e os critérios que deverão ser considerados (RATIONAL, 2006).

Responsável: Gerente de Projeto.

Os artefatos gerados durante a fase de elaboração foram Plano de Desenvolvimento de *Software* (APÊNDICE A), Plano de Iteração da Construção (APÊNDICE F), Especificação Complementar (APÊNDICE B), Documento de Arquitetura do *Software* (APÊNDICE E), Especificação dos Casos de Uso (APÊNDICE D), Especificação da Realização dos Casos de Uso (APÊNDICE G), Especificação dos Requisitos de *Software* (APÊNDICE C), Plano de Teste (APÊNDICE I).

### 3.3 Construção

O objetivo da fase de Construção é minimizar os custos de desenvolvimento, evitando o retrabalho, conseguir um produto de boa qualidade e eficiente. Além disso, todo o sistema deve ser implementado e integrado a um produto de *software*, a fim de produzir versões o mais rápido possível. Enquanto as fases de Concepção e Elaboração têm um perfil de pesquisa, esta fase é focada na produção de *software* em escala (RATIONAL, 2006).



**Figura 13. Workflow Fase de Construção**

Fonte: Adaptado da Rational (2006).

As tarefas executadas na fase de Construção foram:

### **Preparar o Ambiente para a Iteração**

- Verificar Configuração e Instalação de Ferramentas

Selecionar e adquirir as ferramentas necessárias para a realização do projeto (RATIONAL, 2006).

Responsável: Especialista em Ferramentas.

### **Monitorar e Controlar o Projeto**

- Monitorar a Situação do Projeto

Verificar a situação atual do projeto (RATIONAL, 2006).

Responsável: Gerente de Projeto.

- Planejamento e Designação do Trabalho

Acomodar mudanças aprovadas (defeitos, realces), ao produto e aos processos, que se levantam durante uma iteração (RATIONAL, 2006).

Responsável: Gerente de Projeto.

- Lidar com Exceções e Problemas

Iniciar ações corretivas apropriadas aos problemas e exceções, que são levantadas no projeto (RATIONAL, 2006).

Responsável: Gerente de Projeto.

### **Gerenciar Mudanças de Requisitos**

- Revisar os Requisitos

Verificar formalmente o resultado dos requisitos conforme a customização do sistema (RATIONAL, 2006).

Responsável: Revisor Técnico.

- Estruturar os Modelos de Casos de Uso

Extrair o comportamento dos casos de uso que necessitam ser considerados como casos de uso abstratos, que devem ser tratados em iterações anteriores.

Encontrar os novos atores abstratos que definem os papéis que são compartilhados por diversos atores (RATIONAL, 2006).

Responsável: Analista de Sistema.

- Gerenciar as Dependências

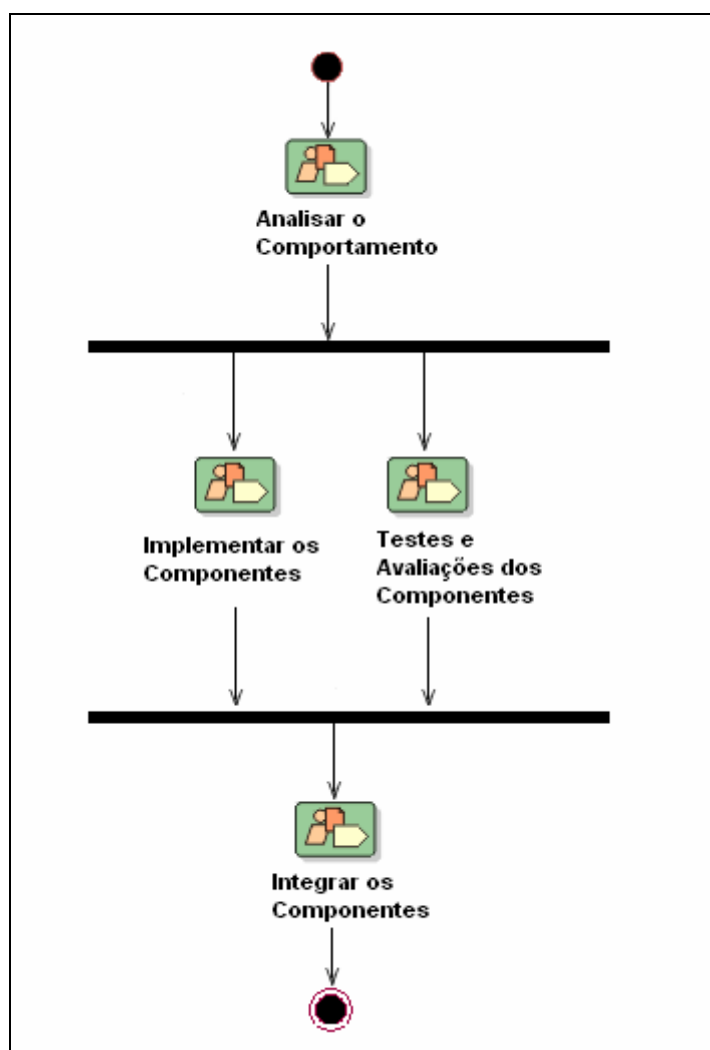
Usar atributos e características dos requisitos do projeto para auxiliar no gerenciamento do escopo e nas mudanças de requisitos que envolvem o projeto (RATIONAL, 2006).



Responsável: Analista de Sistema.

## Implementar Componentes

A tarefa implementar componentes envolve uma série de outras tarefas que são necessárias para realizar o projeto de componentes de um sistema. Como a proposta de solução envolve o uso de um componente, este item é descrito na figura a seguir.



**Figura 14. Workflow da Tarefa Implementar Componentes**  
Fonte: Adaptado da Rational (2006).

As atividades que envolvem o projeto de componentes são descritas a seguir conforme cada tarefa.

### Analisar o Comportamento

- Analisar os Casos de Uso

Identificar as classes que executarão os fluxos de eventos dos casos de uso, distribuir o comportamento dos casos de uso para essas classes, usando a realização dos casos de uso. Identificar as responsabilidades, atributos e associações das classes (RATIONAL, 2006).

Responsável: Projetista.

- Identificar os Elementos do Projeto

Analisar as interações da classe de análise para identificar o elemento modelo do projeto (RATIONAL, 2006).

Responsável: Arquiteto de *Software*.

- Projetar as Interfaces com o Usuário

Produzir um projeto de interface com o usuário que apresente uma explicação sobre o funcionamento do sistema (RATIONAL, 2006).

Responsável: Projetista.

### Implementar os Componentes

- Implementar o Projeto dos Elementos

Produzir uma implementação para a parte do projeto (tal como uma classe do projeto, subsistema do projeto, ou as realizações de caso de uso), ou para reparar um ou mais defeitos. O resultado é o código-fonte, ou atualizações do mesmo (RATIONAL, 2006).

Responsável: Programador.

- Analisar o comportamento da aplicação em execução

Entender o comportamento de um componente durante sua execução. Identificar o comportamento anormal e algumas das ações corretivas que foram solicitadas (RATIONAL, 2006).

Responsável: Programador.

- Executar os Elementos de Testabilidade

Executar funcionalidades especializadas para suportar os requisitos de teste específicos (RATIONAL, 2006).

Responsável: Programador.

- Implementar Testes

Implementar um ou mais testes que permitem a validação dos componentes de *software* de forma individual com a execução física. Desenvolver os testes que podem ser executados conjuntamente com outros testes como parte de uma infra-estrutura maior de teste (RATIONAL, 2006).

Responsável: Programador.

- Executar Testes

Executar a especificação de uma unidade. Verificar a estrutura interna de uma unidade (RATIONAL, 2006).

Responsável: Programador.

- Rever Código

Verificar a Implementação (RATIONAL, 2006).

Responsável: Programador.

- Planejar a Integração do Subsistema

Planejar a ordem em que os elementos contidos no subsistema devem ser integrados (RATIONAL, 2006).

Responsável: Programador.

### Testes e Avaliações dos Componentes

- Implementar o Conjunto de Testes

Montar um conjunto de testes a serem executados, para capturar a situação do sistema. Facilitar a largura e a profundidade apropriadas da cobertura dos testes, exercitando combinações interessantes destes (RATIONAL, 2006).

Responsável: Programador.

- Executar o Conjunto de Testes

Executar o conjunto apropriado de testes requeridos, avaliando a qualidade do sistema. Capturar os resultados destes testes para facilitar avaliações (RATIONAL, 2006).

Responsável: Programador.

#### Integrar os Componentes

- Integrar os Subsistemas

Integrar os elementos em um subsistema. Realizar a integração do subsistema ao sistema (RATIONAL, 2006).

Responsável: Projetista.

### **Testes e avaliações**

- Executar os Testes

Criar e executar um ou mais artefatos de teste, que permitam a validação do produto de *software* com a execução física. Desenvolver os testes que podem ser executados conjuntamente com outros testes como parte de uma infraestrutura maior de teste (RATIONAL, 2006).

Responsável: Programador.

### **Próximo Plano de Iteração**

- Desenvolver Plano de Iteração

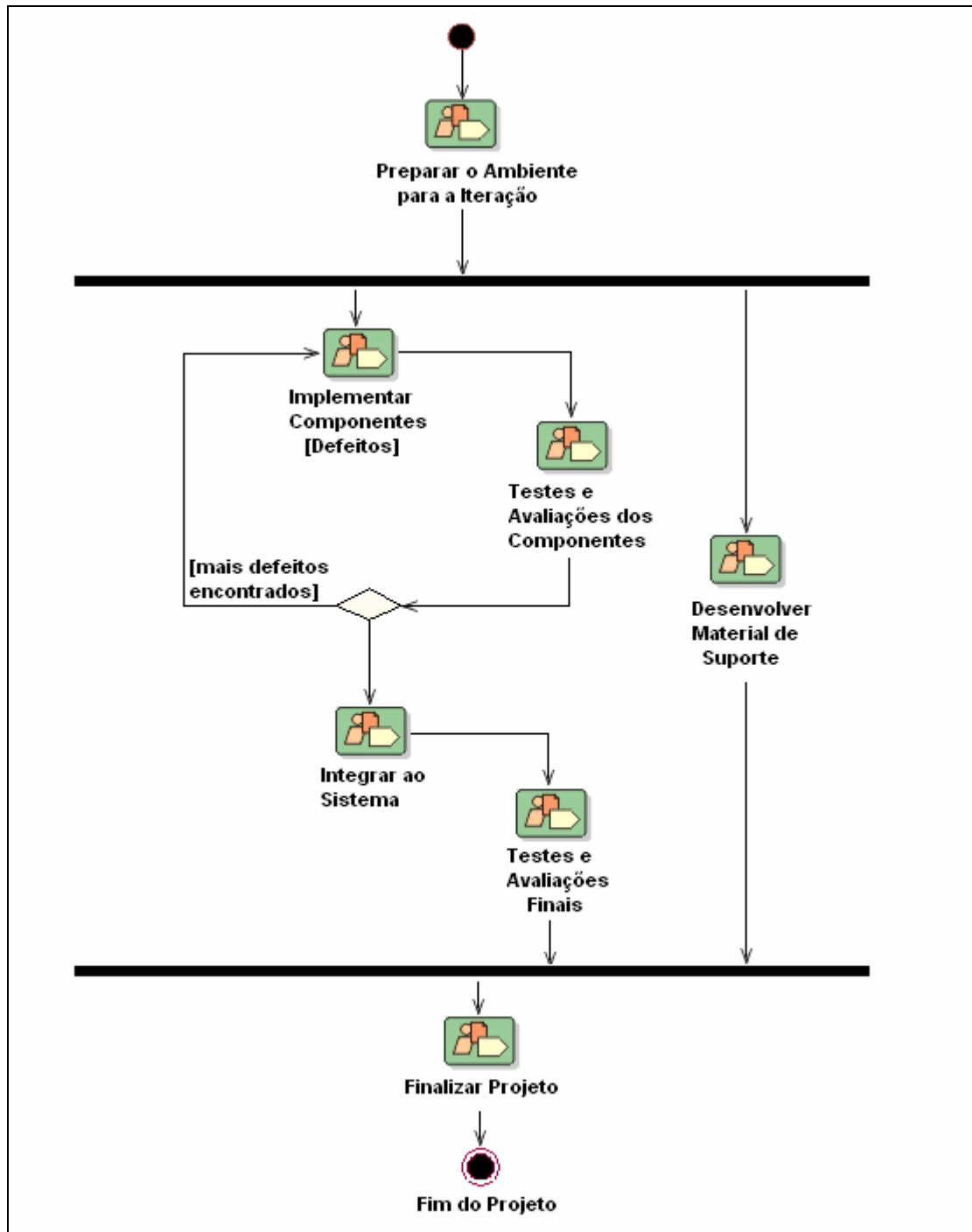
Criar e aprovar as metas propostas para a iteração da próxima fase (RATIONAL, 2006).

Responsável: Gerente de Revisão.

Os artefatos gerados durante a fase de construção foram Plano de Iteração Transição (APÊNDICE F), Plano de Desenvolvimento de *Software* (APÊNDICE A), Especificação dos Casos de Uso (APÊNDICE D), Plano de Implementação (APÊNDICE H), Plano de Teste (APÊNDICE I).

### **3.4 Transição**

O objetivo da fase de Transição é instalar o *software* no ambiente do usuário, executar beta testes para validar o sistema em relação às suas expectativas e treinar os usuários. Neste momento do ciclo de vida, o usuário deve observar principalmente os ajustes do produto configurado, instalado e também a questão de usabilidade. Todas as questões estruturais já foram tratadas nos ciclos de vida anteriores (RATIONAL, 2006).



**Figura 15. Workflow Fase de Transição**  
 Fonte: Adaptado da Rational (2006).

As tarefas executadas na fase de Transição foram:

### **Preparar o Ambiente para a Iteração**

- Verificar Configuração e Instalação de Ferramentas

Selecionar e adquirir as ferramentas necessárias para a realização do projeto (RATIONAL, 2006).

Responsável: Especialista em Ferramentas.

### **Implementar Componentes (Defeitos)**

- Rever Código

Verificar a implementação e execução do sistema (RATIONAL, 2006).

Responsável: Programador.

- Analisar o comportamento da aplicação em execução

Entender o comportamento de um componente durante sua execução.

Identificar o comportamento anormal e algumas das ações corretivas que foram solicitadas (RATIONAL, 2006).

Responsável: Programador.

- Executar os Elementos de Testabilidade

Executar funcionalidades especializadas para suportar os requisitos de testes específicos (RATIONAL, 2006).

Responsável: Programador.

- Implementar Testes

Implementar um ou mais testes que permitem a validação dos componentes de *software* de forma individual com a execução física. Desenvolver os testes que podem ser executados conjuntamente com outros testes como parte de uma infra-estrutura maior de teste (RATIONAL, 2006).

Responsável: Programador.

- Executar Testes

Executar a especificação de uma unidade. Verificar a estrutura interna de uma unidade (RATIONAL, 2006).

Responsável: Programador.

- Planejar a Integração do Subsistema

Planejar a ordem em que os elementos contidos no subsistema devem ser integrados (RATIONAL, 2006).

Responsável: Programador.

### **Teste e Avaliações dos Componentes**

- Executar os Testes

Executar um ou mais artefatos de teste, que permitam a validação dos componentes no *software*. Desenvolver os testes que podem ser executados com um ou mais componentes, auxiliando no teste de uma infra-estrutura maior do sistema (RATIONAL, 2006).

Responsável: Verificador.

### **Integrar ao Sistema**

- Integrar Sistema

Integrar a implementação das partes dos subsistemas no projeto (RATIONAL, 2006).

Responsável: Projetista.

### **Testes e Avaliações Finais**

- Executar os Testes

Criar e executar um ou mais artefatos de teste, que permitam a validação final do produto de *software* com a execução física. Desenvolver os testes que podem ser executados conjuntamente com outros testes como parte de uma infra-estrutura maior de teste (RATIONAL, 2006).

Responsável: Verificador.

### **Desenvolver Material de Suporte**

- Desenvolver Materiais de Suporte

Desenvolver o material de sustentação do usuário final (RATIONAL, 2006).



Responsável: Escritor Técnico.

### **Finalizar Projeto**

- Preparar para Finalizar Projeto

Terminar os formulários de aceitação do projeto, atribuir novas equipes de funcionários do projeto e transferir outros recursos do projeto (RATIONAL, 2006).

Responsável: Gerente de Projeto.

Os artefatos gerados durante a fase de transição foram Plano de Desenvolvimento de *Software* (APÊNDICE A), Plano de Implementação (APÊNDICE H), Plano de Teste (APÊNDICE I).

## **4 DESENVOLVIMENTO**

O capítulo descreve o ambiente tecnológico de desenvolvimento e os procedimentos utilizados para a codificação do sistema e do componente.

### **4.1 Ambiente de Desenvolvimento**

O ambiente de desenvolvimento envolve as tecnologias que foram instaladas e configuradas para o desenvolvimento do trabalho. A Figura 16 ilustra o cenário dessas tecnologias e como elas interagem entre si.

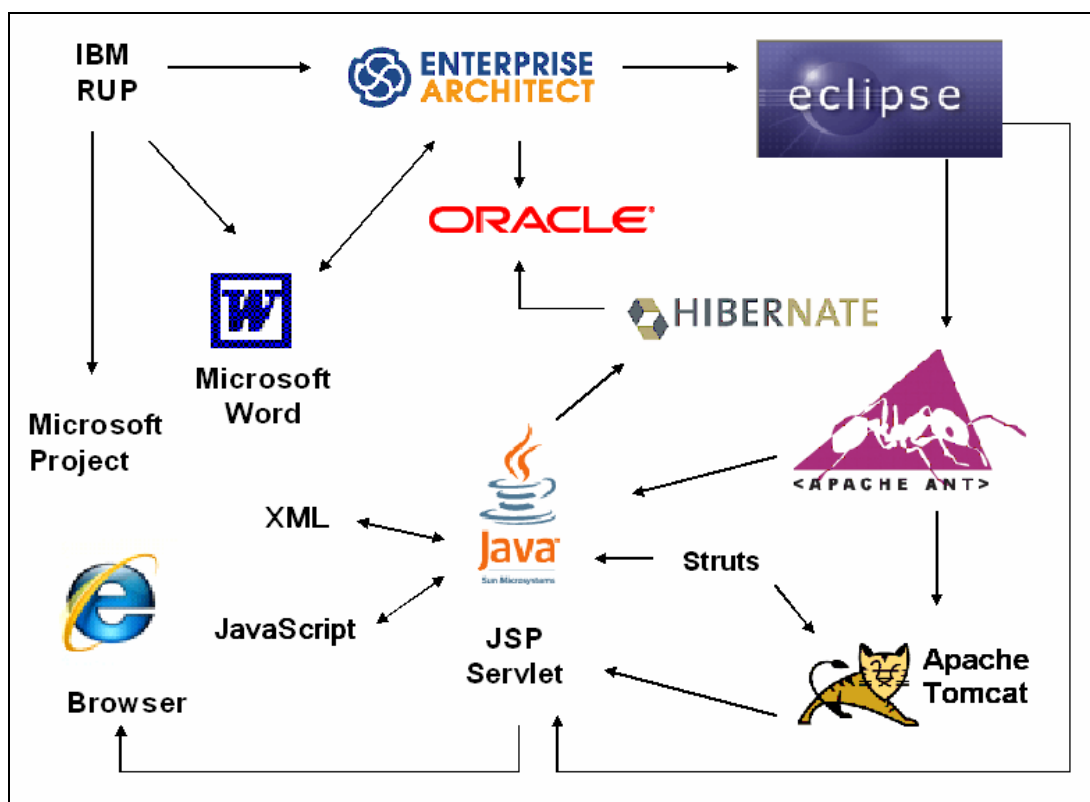


Figura 16. Cenário Tecnológico

A tecnologia IBM RUP foi considerada a linha-mestra do desenvolvimento do projeto de *software*. É um processo proprietário de engenharia de *software* criado pela *Rational Software Corporation*, adquirida pela IBM tornando-se uma *brand* na área de *software*, fornecendo técnicas a serem seguidas pelos membros da equipe de desenvolvimento de *software* com o objetivo de aumentar a sua produtividade. Os artefatos para documentação, disponibilizados pelo IBM RUP foram capturados e editados no editor de texto Microsoft Word.

O *Enterprise Architect* foi a ferramenta de modelagem que permitiu o desenvolvimento da modelagem do sistema e do componente, comprovando a coerência existente entre a modelagem e o desenvolvimento.

Em relação às outras tecnologias, é feita uma breve descrição:

- *Apache Tomcat*: é um servidor de aplicações Java para *web*. É distribuído como *software* livre e desenvolvido como código aberto dentro do conceituado projeto *Apache Jakarta* e oficialmente endossado pela Sun como a

Implementação de Referência (RI) para as tecnologias *Java Servlet* e *JavaServer Pages* (JSP). O *Tomcat* é robusto e eficiente o suficiente para ser utilizado em um ambiente de produção (TOMCAT, 2006).

- *Oracle*: Banco de dados relacional compatível com o padrão SQL-ANSI-92. O acesso é feito via JDBC, independente de plataforma. Para poder utilizá-lo é necessário comprar licença.
- *Eclipse*: é uma IDE *open Source* distribuída como um componente principal (o Eclipse SDK) com o ambiente básico, com os recursos fundamentais para desenvolvimento Java padrão, mais diversos componentes adicionais distribuídos na forma de *plug-ins*. Estes *plug-ins* estendem a funcionalidade do ambiente e acrescentam suporte a recursos e tecnologias específicos (ECLIPSE, 2006).
- *Hibernate*: é um *framework* de acesso a banco de dados escrito em Java. Ele é um *software* livre de código aberto distribuído. O objetivo do *Hibernate* é facilitar a construção de aplicações Java dependentes de bases de dados relacionais, particularmente, facilitar o desenvolvimento das consultas e atualizações dos dados. O uso de ferramentas de mapeamento objeto relacional, como o *Hibernate*, diminuem a complexidade resultante da convivência de modelos diferentes; o modelo orientado a objetos (da linguagem Java) e o relacional (da maioria dos SGBDs) (HIBERNATE, 2006).
- *Apache Ant*: é uma ferramenta utilizada para automatizar a construção de *software*. Ela é escrita na linguagem Java e foi desenvolvida inicialmente para ser utilizada em projetos desta linguagem. O *Ant* utiliza um arquivo no formato XML para descrever o processo de construção (*build*) e suas dependências. Por padrão este arquivo XML tem o nome “build.xml”. A ferramenta *Ant* é um

projeto da *Apache Software Foundation*. É um *software* livre, licenciado sob a licença *Apache* (ANT, 2006).

- *Java Server Pages* (JSP): é uma tecnologia utilizada no desenvolvimento de aplicações para *web*. Por ser baseada na linguagem de programação Java ela tem a vantagem da portabilidade de plataforma, que permite a sua execução em outros sistemas operacionais. Esta tecnologia permite ao desenvolvedor de páginas para Internet produzir aplicações que, acessam o banco de dados, manipulam arquivos no formato texto, captam de informações a partir de formulários e captam informações sobre o visitante e sobre o servidor (JAVA, 2006).
- XML: é um subtipo de SGML (*Standard Generalized Markup Language* - Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet (XML, 2006).
- *Servlet*: é um componente que disponibiliza ao programador da linguagem Java uma interface para o servidor *web* (ou servidor de aplicação), através de uma API. As aplicações baseadas no *Servlet* geram conteúdo dinâmico (normalmente HTML) e interagem com os clientes, utilizando o modelo *request/response*. Os *servlets* normalmente utilizam o protocolo HTTP, apesar de não serem restritos a ele. Um *Servlet* necessita de um *container web* para ser executado (JAVA, 2006).
- *Struts*: é um *framework* de desenvolvimento da camada controladora, em uma estrutura seguindo o padrão *Model 2* (uma variante do MVC oficializada pela Sun), de aplicações *web* (principalmente) construído em Java para ser utilizado em um *container web* em um servidor J2EE. Este *framework* foi originalmente

desenvolvido por Ted Husted e doado para a *Apache Software Foundation*, onde continua sendo desenvolvido segundo o padrão desta fundação (STRUTS, 2006).

- *JavaScript*: é uma linguagem de programação criada pela Netscape em 1995, que a princípio se chamava LiveScript, para atender, principalmente as validação de formulários no lado cliente (programa navegador), interação com a página. Assim, foi feita como uma linguagem de script. Javascript tem sintaxe semelhante a do Java, mas é totalmente diferente no conceito e no uso.
- *Java*: é uma linguagem de programação orientada a objeto desenvolvida na década de 90 pelo programador James Gosling, na empresa Sun Microsystems. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual (JAVA, 2006).
- *Microsoft®Internet Explorer* (IEExplorer): é um navegador de licença proprietária produzido inicialmente pela Microsoft em 23 de agosto de 1995. É de longe o navegador mais usado atualmente uma vez que é distribuído em cada versão do sistema operacional Windows (MICROSOFT, 2006).

## 4.2 Implementação do Sistema Manutenção

A implementação do sistema foi realizada seguindo o processo de desenvolvimento do RUP e utilizando os conceitos de modularização. A seqüência de operações têm como fluxo básico a consulta e a partir desta a realização de cadastro, alteração

e exclusão. Os fluxos de eventos de cada transação estão descritos no apêndice D. A Figura 17 a seguir demonstra a tela de consulta:

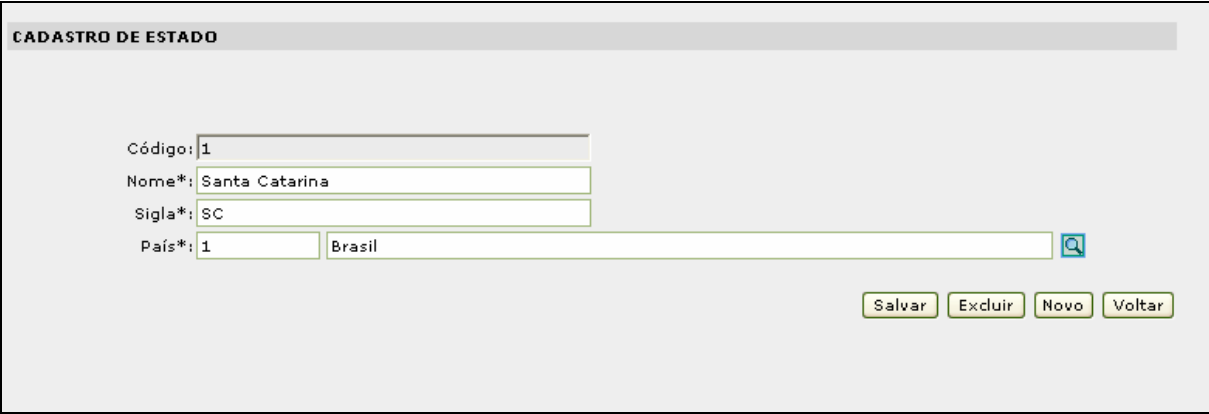
Código	Nome	Sigla	País	
1	Santa Catarina	SC	Brasil	
2	São Paulo	SP	Brasil	
3	Rio de Janeiro	RJ	Brasil	

**Figura 17. Tela de Consulta**

Para realizar o cadastro de uma nova informação, é necessário clicar no botão “Novo” visualizado na Figura 17, para então ser aberta a tela demonstrada na figura a seguir:

**Figura 18. Tela de Cadastro**

Já para a alteração e/ou exclusão das informações cadastradas no sistema é necessário selecionar um item, resultado da consulta, através de um clique na imagem para que esta redirecione para a página de cadastro preenchida. Assim, a alteração das informações é feita quando o usuário clica no botão “Salvar” e a exclusão quando ele clica no botão “Excluir”. A Figura 19 ilustra essa operação:



A interface de usuário para a edição de um registro no sistema 'CADASTRO DE ESTADO'. O formulário contém os seguintes campos: 'Código:' com o valor '1', 'Nome\*:' com o valor 'Santa Catarina', 'Sigla\*:' com o valor 'SC', e 'País\*:' com o valor '1' e um campo de texto adjacente contendo 'Brasil'. Um ícone de lupa está à direita do campo de país. Na parte inferior direita, há quatro botões: 'Salvar', 'Excluir', 'Novo' e 'Voltar'.

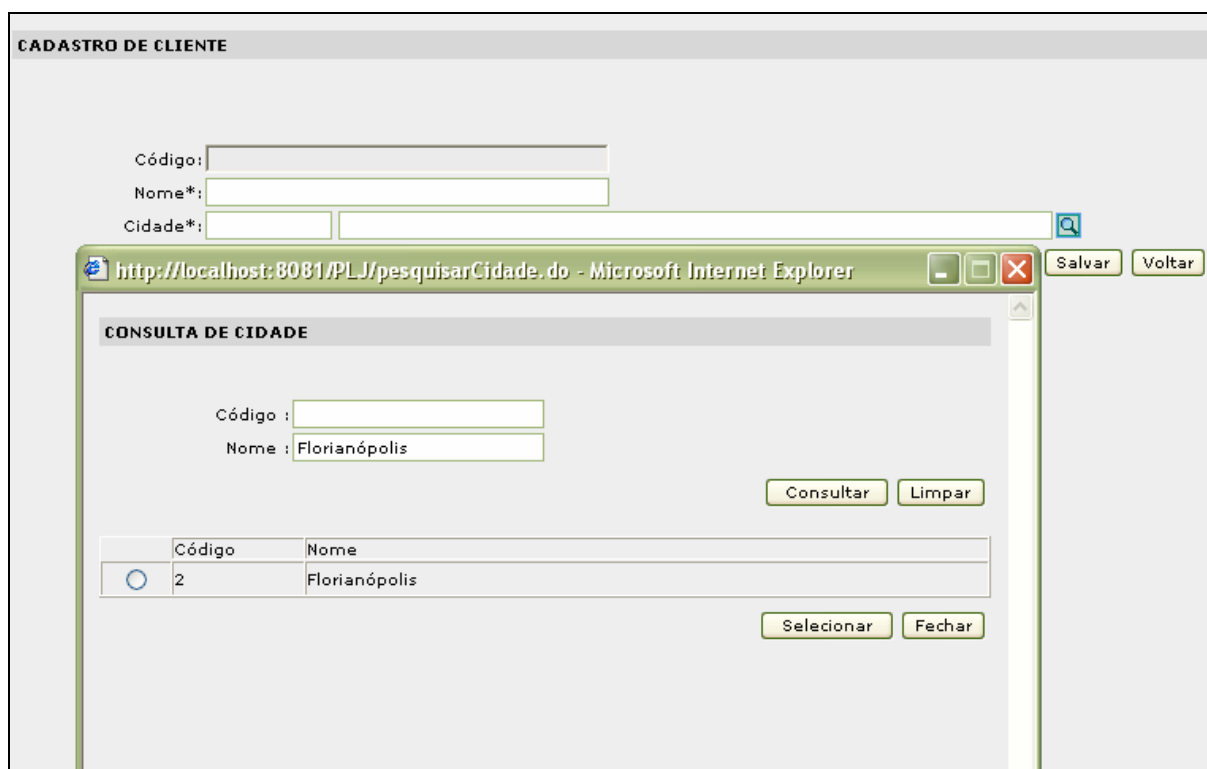
**Figura 19. Tela de Edição**

Para implementação fez-se uso das ferramentas Eclipse, Internet Explorer e SQL-Plus. O Eclipse foi utilizado para escrever o código-fonte do sistema. Como o sistema é *web* utilizou-se o Internet Explorer como navegador para fazer as transações de consulta, cadastro, alteração e exclusão. O SQL-Plus foi utilizado para executar os *scripts* do banco de dados para a criação das tabelas e *scripts* de consulta às tabelas criadas.

As tecnologias utilizadas no desenvolvimento do sistema foram *Tomcat* como servidor da aplicação, *Java* como linguagem de programação, o *Framework Struts* para auxiliar na geração das interfaces e ações da aplicação *web*, o JavaScript para fazer algumas validações, JSP para criar as páginas *web*, o *hibernate* para fazer a persistência dos dados com o banco de dados e os CSS para criar o estilo das páginas do sistema.

O sistema possui uma tela de consulta para dados que tem um relacionamento com outra tabela. Por exemplo, temos o cadastro de cliente que mora em uma cidade. Essa cidade é cadastrada no banco de dados em uma tabela independente de cliente. O objetivo da tela é buscar através de uma consulta as cidades cadastradas para que seja selecionada aquela que é necessário para o cadastro. A Figura 20 demonstra o fluxo:





**Figura 20. Tela de Consulta Cidade para o Cliente**

Na análise realizada na arquitetura descrita no APÊNDICE E, verificou-se que a funcionalidade de consulta está bem definida a esse módulo pode ser separado do sistema tornando-se um componente. Foi então desenvolvido um componente de consulta e o mesmo é descrito a seguir.

### 4.3 Implementação do Componente LJ

Com o intuito de demonstrar a reutilização criou-se um componente de consulta genérico. Sabendo que a funcionalidade de consulta é realizada em diversas partes do sistema, a redundância e o tempo de programação aumentam significativamente dependendo do tamanho do sistema, pois quanto mais tabelas, mais consultas irão existir. Dessa forma, com o

auxílio de um componente que execute essa funcionalidade, a redundância é descartada e o tempo de implementação e entrega do produto são reduzidos.

Na implementação do componente fez-se uso das ferramentas: Eclipse, Apache Ant, Internet Explorer e SQL-Plus. O Eclipse foi utilizado para escrever o código-fonte. Já o Apache Ant para toda a distribuição do desenvolvimento, tendo em vista que o componente será utilizado na *web*. A visualização foi feita no Internet Explorer. Toda a parte de conectividade e verificação do banco de dados foi feita com o auxílio do SQL-Plus.

As tecnologias utilizadas foram *Tomcat* como servidor da aplicação, Java como linguagem de programação, *Servlets* para criar as páginas dinâmicas para *web*, XML para criação do arquivo de definição das consultas.

O desenvolvimento foi constituído por quatro etapas principais:

- Criação do XML e DTD;
- Criação das Classes do Componente;
- Criação da Página para Consulta.

A seguir descreve-se com mais detalhes cada uma dessas etapas.

#### 4.3.1 Criação do XML e DTD

Com o auxílio de um documento XML e suas *tags* de marcação iniciou-se a delimitação das pesquisas <PESQUISAS>, procurando demarcar o esquema de cada pesquisa <PESQUISA>. Cada pesquisa contém o nome de sua tabela no banco de dados <NOME>, um ou mais filtros a serem exibidos na tela de consulta <FILTRO>, sendo que cada filtro tem <TIPO>, <CAMPO\_PESQUISA> e <ROTULO\_PESQUISA>. A *tag* <TIPO> serve para

indicar qual tipo de campo será consultado (“String” ou “int”). Por sua vez, a *tag* <CAMPO\_PESQUISA> indica o atributo da tabela do banco de dados que se deseja pesquisar e a *tag* <ROTULO\_PESQUISA> indica o rótulo de filtro que o usuário deseja exibir na tela no momento da consulta.

Os campos a serem exibidos na tela no momento da listagem em cada consulta serão indicados pela *tag* <EXIBIR> que é delimitada para cada coluna de listagem <COLUNAS>, na qual foi especificado seu <CAMPO> atributo da tabela no banco de dados para exibir os valores retornados. A *tag* <ROTULO> indica a descrição da coluna no momento da listagem.

Contudo, antes de fechar a *tag* <PESQUISA> com </PESQUISA> no documento XML, é utilizada a *tag* <CHAVE> para passar o atributo chave da tabela contida na *tag* <NOME> e a *tag* <DESCRICAO\_CHAVE> para passar o campo descrição da mesma tabela.

A seguir uma ilustração do XML, mostrando a formatação para consulta de um Cliente:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PESQUISAS SYSTEM "pesquisas.dtd">
<PESQUISAS>
  <PESQUISA>
    <NOME>CLIENTE</NOME>
    <FILTRO>
      <TIPO>int</TIPO>
      <CAMPO_PESQUISA>CLIENTE_ID</CAMPO_PESQUISA>
      <ROTULO_PESQUISA>Cliente Id</ROTULO_PESQUISA>
    </FILTRO>
    <FILTRO>
      <TIPO>String</TIPO>
      <CAMPO_PESQUISA>NOME_CLIENTE</CAMPO_PESQUISA>
      <ROTULO_PESQUISA>Nome</ROTULO_PESQUISA>
    </FILTRO>
    <EXIBIR>
      <COLUNAS>
        <CAMPO>CLIENTE_ID</CAMPO>
        <ROTULO>Cliente Id</ROTULO>
      </COLUNAS>
      <COLUNAS>
        <CAMPO>NOME_CLIENTE</CAMPO>
        <ROTULO>Nome</ROTULO>
      </COLUNAS>
    </EXIBIR>
    <CHAVE>CLIENTE_ID</CHAVE>
    <DESCRICAO_CHAVE>NOME_CLIENTE</DESCRICAO_CHAVE>
  </PESQUISA>

```

**Figura 21. Documento XML para Construção da Pesquisa**

Especificou-se a Definição de Tipo de Documento (DTD), que serve para especificar quais elementos ou atributos são permitidos no documento XML e em que local do documento eles podem aparecer. Podemos definir, então, que o DTD é uma forma de validar o documento XML. A Figura 22 demonstra o DTD criado para validar as pesquisas.

```

<!ELEMENT PESQUISAS (PESQUISA+, CONEXAO) >

<!ELEMENT PESQUISA (NOME, FILTRO+, EXIBIR, CHAVE, DESCRICAO_CHAVE) >
<!ELEMENT CONEXAO (USUARIO, SENHA, DRIVER, CONN) >

<!ELEMENT NOME (#PCDATA) >
<!ELEMENT FILTRO (TIPO, CAMPO_PESQUISA, ROTULO_PESQUISA) >
<!ELEMENT EXIBIR (COLUNAS+) >
<!ELEMENT CHAVE (#PCDATA) >
<!ELEMENT DESCRICAO_CHAVE (#PCDATA) >

<!ELEMENT TIPO (#PCDATA) >
<!ELEMENT CAMPO_PESQUISA (#PCDATA) >
<!ELEMENT ROTULO_PESQUISA (#PCDATA) >

<!ELEMENT COLUNAS (CAMPO, ROTULO) >

<!ELEMENT CAMPO (#PCDATA) >
<!ELEMENT ROTULO (#PCDATA) >

<!ELEMENT USUARIO (#PCDATA) >
<!ELEMENT SENHA (#PCDATA) >
<!ELEMENT DRIVER (#PCDATA) >
<!ELEMENT CONN (#PCDATA) >

```

**Figura 22. DTD de Validação para as Pesquisas**

Para facilitar a configuração da conexão ao banco de dados criou-se ainda, no documento XML a tag <CONEXAO> que contém o usuário <USUARIO>, a senha <SENHA>, o *driver* <DRIVER> e o *host* de conexão <CONN>.

```

</PESQUISA>

<CONEXAO>
  <USUARIO>admin</USUARIO>
  <SENHA>admin</SENHA>
  <DRIVER>oracle.jdbc.driver.OracleDriver</DRIVER>
  <CONN>jdbc:oracle:thin:@localhost:1521:Oracle</CONN>
</CONEXAO>

</PESQUISAS>

```

**Figura 23. Dados da Conexão ao Banco de Dados no XML**

Após a criação e validação do XML, é necessário realizar a leitura e manipulação dos dados nele contidos. Para isso, foram criadas classes que realizam essas funções e outras que são necessárias para o funcionamento correto do componente.

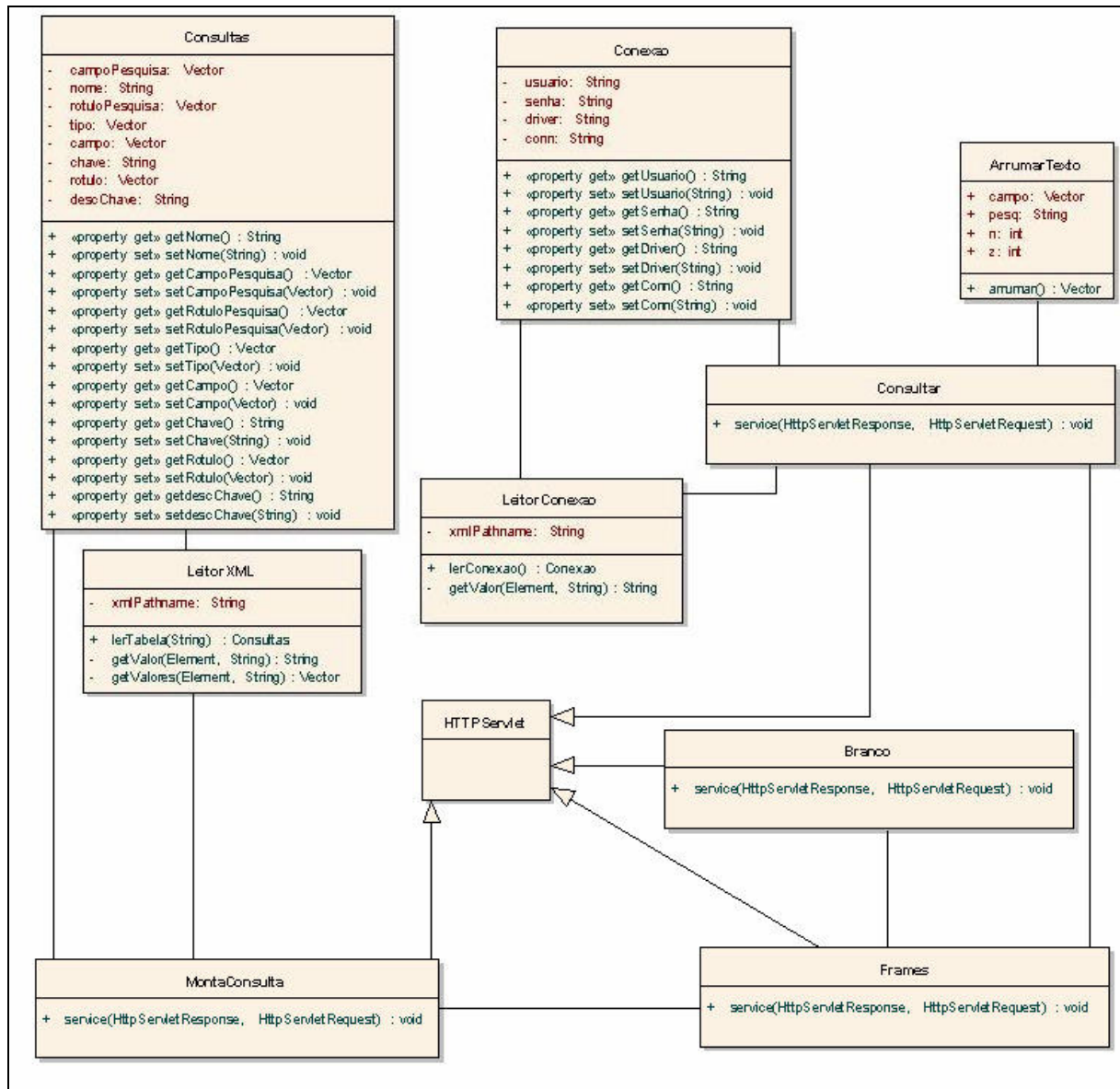
#### 4.3.2 Criação das Classes do Componente

Utilizou-se o pacote `org.w3c.dom` que provê uma interface de programação padrão para aplicações que manipulam XML. O pacote transforma um documento XML em uma estrutura de árvore na memória.

As classes que utilizam esse pacote são: “LeitorXML.java” e “LeitorConexao.java”. Assim, logo após ler o documento XML, a classe “LeitorXML.java” retorna um objeto `Consulta` com o auxílio da classe “Consulta.java”, que contém todos os dados da pesquisa a ser realizada no banco de dados. Já a classe “LeitorConexao.java” retorna um objeto `Conexao` com o auxílio da classe “Conexao.java”, que possui todos os dados da conexão ao banco de dados.

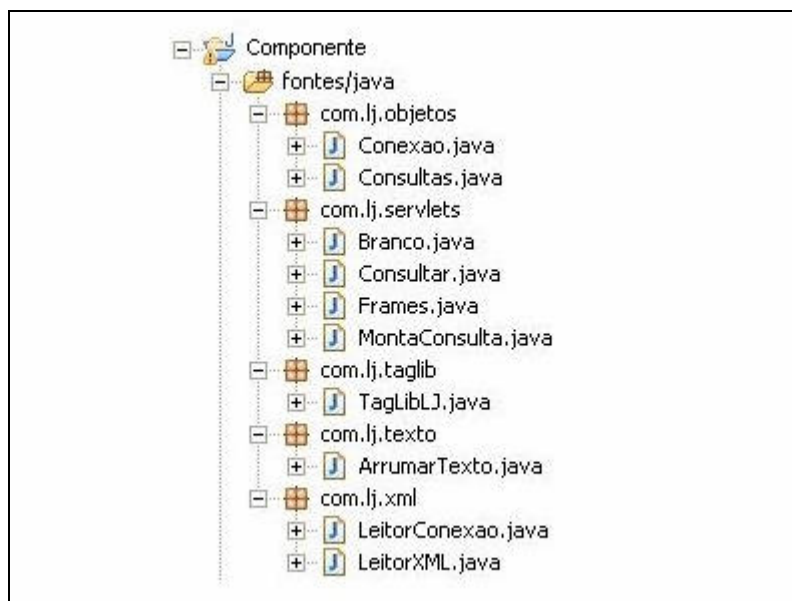
As classes “MontaConsulta.java”, “Frames.java”, “Branco.java” e “Consultar.java” estendem `HttpServlet` e são utilizadas para criar a página de consulta.

O diagrama de classes utilizado para desenvolver o componente está representado na Figura 24.



**Figura 24. Diagrama de Classes do Componente**

Para realizar a implementação das classes, criou-se uma estrutura de pacotes. Isso foi feito para separar as classes que possuem funcionalidades semelhantes. Essa estrutura está representada na Figura 25:



**Figura 25. Pacotes do Componente**

#### 4.3.3 Criação da Página para Consulta

O componente de consulta é invocado com a utilização de uma *taglib*. A classe “TagLibLJ.java” gera um campo para descrição do valor que é retornado e um botão para “chamar” a devida consulta, fazendo a interface entre página requisitante e o componente de consulta. Foi implementada uma TLD para a classe “TagLib.java”, no qual foram criados os parâmetros que são necessários para a execução do componente. A Figura 26 mostra a TLD gerada para o componente.



```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.2</jspversion>
  <shortname></shortname>
  <uri>tagLibLJ</uri>
  <tag>
    <name>ljpesquisa</name>
    <tagclass>com.lj.taglib.TagLibLJ</tagclass>
    <attribute>
      <name>tabela</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>form</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>campo</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>value</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>img</name>
      <required>false</required>
    </attribute>
  </tag>
</taglib>

```

**Figura 26. TLD com os Parâmetros da Taglib**

Os parâmetros da TLD são: o nome da tabela a ser pesquisada, o formulário de retorno, o campo deste formulário para retornar o elemento, nome do botão e se optar por mostrar uma imagem é possível colocar o caminho da mesma para ser localizada dentro do projeto.

Na página a *taglib* é chamada através de uma diretiva, localizada no topo. A partir dessa diretiva é possível utilizar a *tag* <componente: ljpesquisa> para fazer a interface entre o componente e a página requisitante. Dessa forma, com todos os parâmetros preenchidos, a funcionalidade pode ser executada, como mostra a Figura 27.

```

<%@ taglib uri="../../../WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="../../../WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="../../../WEB-INF/struts-logic.tld" prefix="logic"%>
<%@ taglib uri="../../../WEB-INF/TagLibLJ.tld" prefix="componente"%>
<%@include file="/spjInclude.jsp"%>
.
.
<table width="100%" border="0" cellpadding="0" cellspacing="1">
.
.
<tr>
<td align="right"><bean:message key="label.field.estado.paisId"/>: </td>
<td colspan="4">
<table width="100%" cellpadding="0" cellspacing="0" border="0">
<tr>
<td nowrap="nowrap">
<html:text property="paisId" style="width: 80px" readonly="true"/>
<div>
<componente:ljpesquisa
campo="paisId"
form="estadoForm"
tabela="PAIS"
value="Pesquisar">
</componente:ljpesquisa>
</div>
</td>
</tr>
</table>
<html:hidden property="estado.pais.paisId"/>
</td>
</tr>
</table>
.
.

```

Figura 27. Código JSP utilizando a *Taglib* sem o Parâmetro “img”

A página web gerada com o JSP mostrado acima ficaria dessa forma, destacando o campo criado pela *taglib*:

**CONSULTA DE ESTADO**

Nome:

Sigla:

País:




Código	Nome	Sigla	País	
1	Santa Catarina	SC	Brasil	
2	São Paulo	SP	Brasil	
3	Rio de Janeiro	RJ	Brasil	

Figura 28. Página demonstrando a *Taglib* sem o Parâmetro “img”

A *taglib* permite que o usuário tenha a opção de colocar uma imagem de sua preferência no lugar do botão que “chama” a página de pesquisa. Para que isso ocorra é

necessário acrescentar o parâmetro “img” na tag <componente:ljpesquisa> como mostra a Figura 29.

```
<%@ taglib uri="../../../WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="../../../WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="../../../WEB-INF/struts-logic.tld" prefix="logic"%>
<%@ taglib uri="../../../WEB-INF/TagLibLJ.tld" prefix="componente"%>
<%@include file="/spjInclude.jsp"%>
.
.
<table width="100%" border="0" cellpadding="0" cellspacing="1">
.
.
.
<tr>
.
.
.
<td align="right"><bean:message key="label.field.estado.paisId"/>: </td>
<td colspan="4">
<table width="100%" cellpadding="0" cellspacing="0" border="0">
<tr>
<td nowrap="nowrap">
<html:text property="paisId" style="width: 80px" readonly="true"/>

<componente:ljpesquisa
  campo="paisId"
  form="estadoForm"
  tabela="PAIS"
  value="Pesquisar"
  img="imagem/botProcurar.gif">
</componente:ljpesquisa>

</td>
</tr>
</table>
<html:hidden property="estado.pais.paisId"/>
</td>
</tr>
</table>
.
.
.
```


**Figura 29. Código JSP utilizando a Taglib com o Parâmetro “img”**




Dessa forma a visualização da página é alterada, aparecendo uma imagem para fazer a interface. A Figura 30 ilustra essa diferença.

**CONSULTA DE ESTADO**

Nome:

Sigla:

País:   

Código	Nome	Sigla	País	
1	Santa Catarina	SC	Brasil	
2	São Paulo	SP	Brasil	
3	Rio de Janeiro	RJ	Brasil	

**Figura 30. Página demonstrando a Taglib com o Parâmetro “img”**

Internamente, a classe “TaglibLJ.java” gera código *javascript* que realiza a operação de abrir uma nova janela *popup*. Este código possui um método chamado “abrir” que realiza a abertura da janela, além disso, cria um campo para o retorno da descrição da informação selecionada na pesquisa e um botão. Esse trecho gerado é demonstrado na Figura 31.

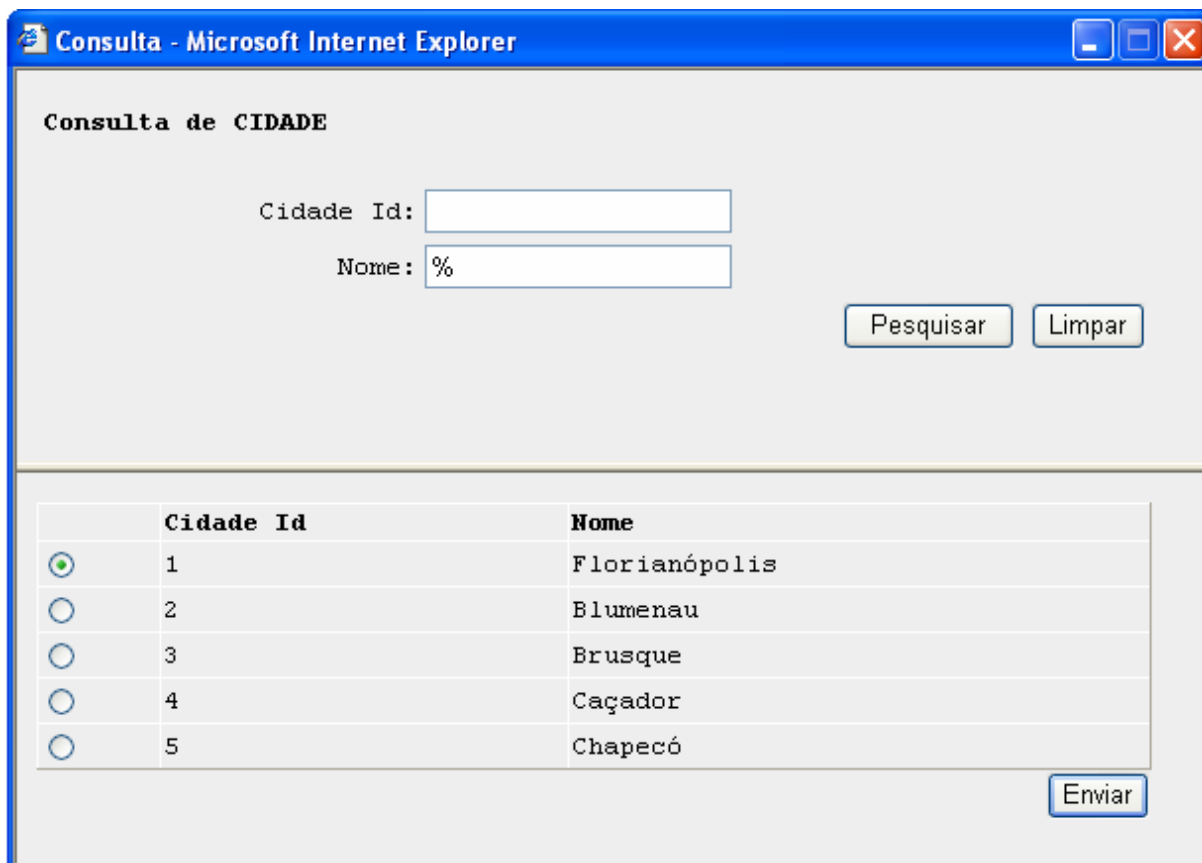
```

*
*
*
<tr>
  <td align="right">
    País:
  </td>
  <td colspan="4">
    <table width="100%" cellpadding="0" cellspacing="0" border="0">
      <tr>
        <td nowrap="nowrap">
          <input type="text" name="paisId" value="" readonly="readonly" style="width: 80px">
          <script type="text/javascript" language="javascript">
            function abrir() {
              window.open("servlet/com.lj.servlets.Frames?
                tabela=PAIS
                &form=estadoForm
                &camp=paisId" ," ,scrollbars=yes,
                width=600,
                height=400);
            }
          </script>
          <input type="text" name="chave_descricao" disabled="disabled" value="">
        </td>
        <td>
          
        </td>
      </tr>
    </table>
    <input type="hidden" name="estado.pais.paisId" value="">
  </td>
</tr>
*
*
*

```

**Figura 31. Trecho gerado pela Taglib**

Após clicar no botão ou imagem abre-se uma página que é montada conforme a tabela passada como parâmetro, a qual possui os campos para filtrar e os botões de “Pesquisar” e “Limpar”. Depois de preencher um filtro e clicar no botão “Pesquisar” a mesma página retornará uma lista de dados relacionados à tabela que está sendo pesquisada. Ao selecionar uma opção e clicar no botão “Enviar” as informações são enviadas à página requisitante e a janela é fechada. A seguir é ilustrada a página de pesquisa gerada pelo componente:



**Consulta de CIDADE**

Cidade Id:

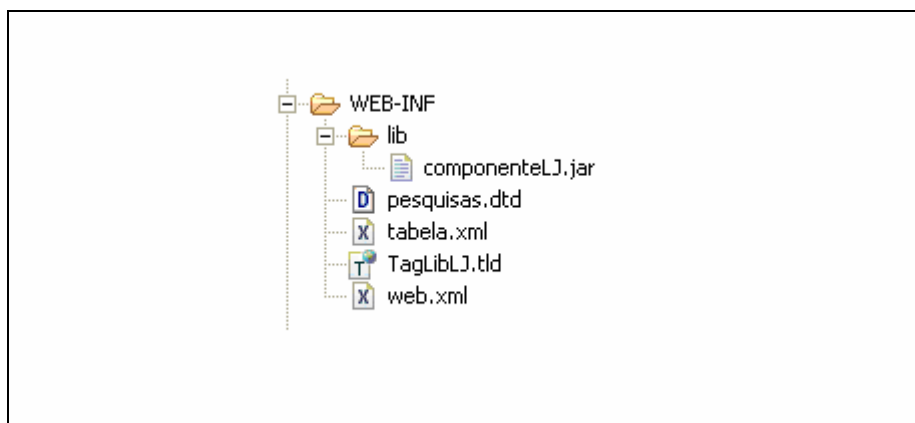
Nome:

	Cidade Id	Nome
<input checked="" type="radio"/>	1	Florianópolis
<input type="radio"/>	2	Blumenau
<input type="radio"/>	3	Brusque
<input type="radio"/>	4	Caçador
<input type="radio"/>	5	Chapecó

**Figura 32. Página de Consulta**

Durante a execução dos procedimentos descritos anteriormente é feito a leitura no XML que resulta no preenchimento do objeto Consulta. Esse objeto contém os itens da pesquisa que são necessários para montar a página com o auxílio do *Servlet*. Os filtros da página, montados a partir da configuração no XML, são utilizados para montar o SQL automaticamente e fazer a consulta ao banco de dados. Da mesma forma, os campos que são retornados da execução do SQL também são configurados, assim como a descrição dos rótulos.

Contudo, o Componente LJ no formato *Java Archive* (JAR) permite que o mesmo seja utilizado em qualquer parte da aplicação. Assim, o componente deve estar localizado dentro do diretório “lib” e seus arquivos de configuração (XML, TLD e DTD) devem estar presentes no diretório “WEB-INF” do projeto que faz uso do componente. A Figura 33 demonstra o local onde devem ser colocados esses arquivos.



**Figura 33. Localização do JAR e respectivos Arquivos de Configuração**

A partir desse momento, com a implementação pronta, testada e documentada, o próximo passo é validar para demonstrar os resultados obtidos.

## 5 VALIDAÇÃO

Este capítulo busca descrever o procedimento utilizado para validar os objetivos do trabalho. Além disso, são descritos os resultados obtidos e uma análise técnica dos objetivos que possuem uma abordagem prática, finalizando com uma recomendação de uso.

Os objetivos mais gerais referentes ao estudo para aprofundar os conhecimentos não foram abordados na validação, pois esses são utilizados durante todo o desenvolvimento do projeto.

Em relação aos objetivos que dizem respeito à análise, projeto e implementação do sistema e do componente, estes são validados a partir dos seguintes critérios: quanto ao código-fonte, coesão, acoplamento, tempo de desenvolvimento, tamanho da aplicação e reutilização.

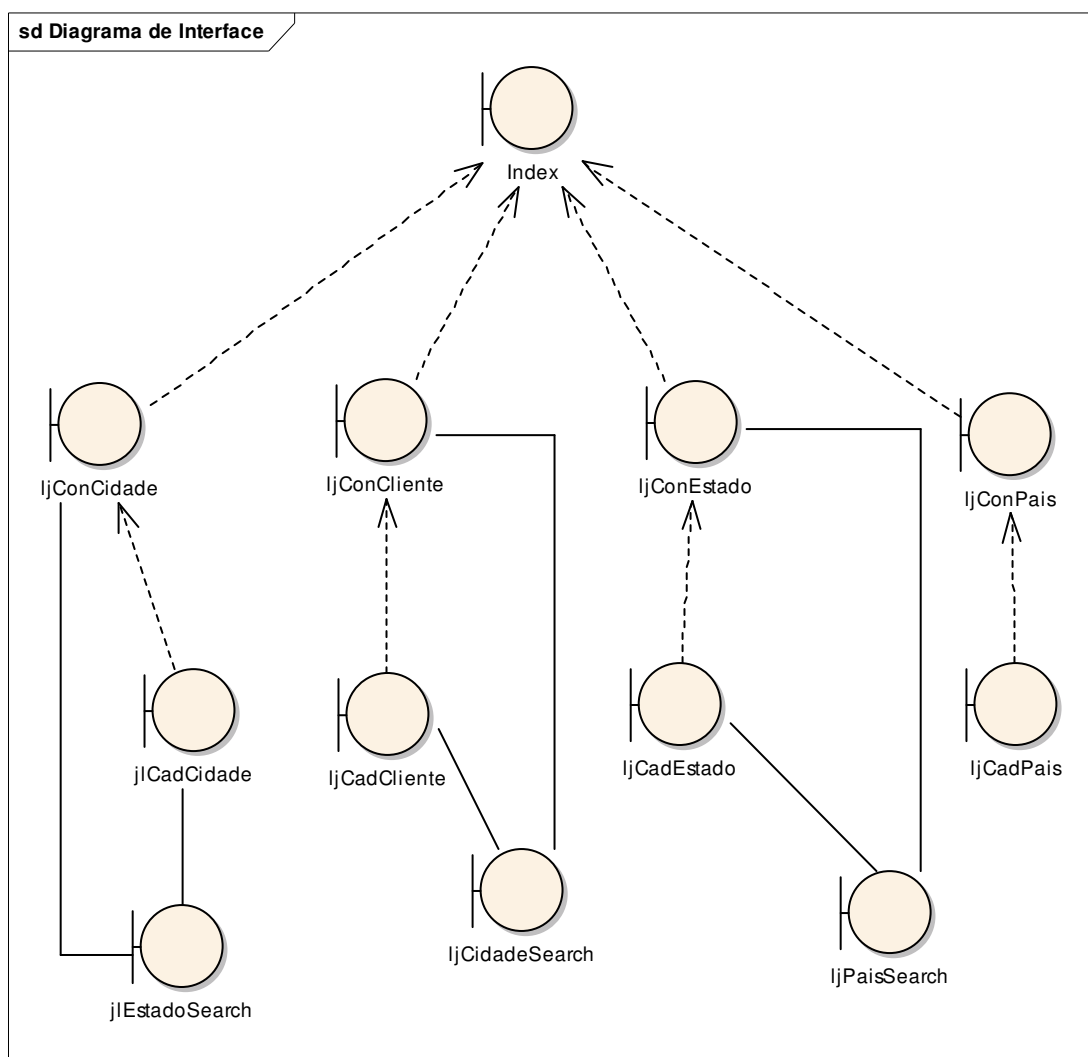
A grande maioria dos sistemas de software faz uso de consultas. Esta funcionalidade é bastante comum e foi utilizada como ponto de referência em toda a validação.

Durante a análise do sistema Manutenção verificou-se a existência de uma série de consultas que se propõe a fazer determinadas regras de negócio com funcionalidades semelhantes. Estas regras de negócio resumem-se em realizar consultas em determinadas tabelas em um banco de dados para preencher um campo que está relacionado com outras



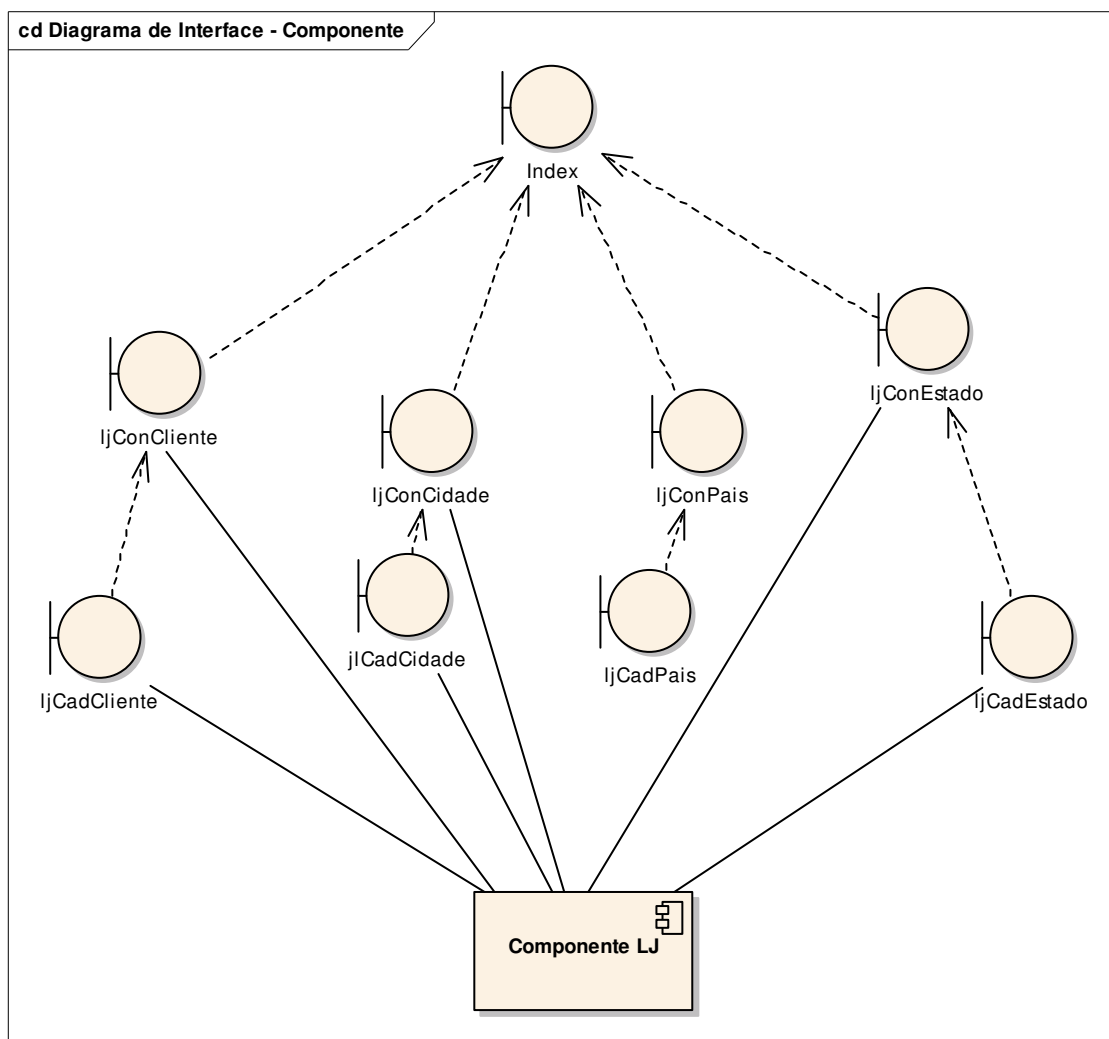


significativa de linhas e isso aconteceu porque o sistema foi desenvolvido com poucas consultas como mostra a Figura 35.



**Figura 35. Diagrama de Interfaces do Sistema Manutenção com as Consultas Internas**

No entanto, se simularmos o uso do componente em um sistema maior e mais complexo, a redução de linhas de código é significativa, pois todas as funcionalidades da consulta são implementadas somente uma vez e não de forma repetitiva como acontece quando a implementação ocorre dentro do sistema. O diagrama de interface abaixo demonstra a utilização do componente no sistema.



**Figura 36. Diagrama de Interfaces Utilizando o Componente para fazer as Consultas**

Para dar uma abordagem científica à esse critério utilizou-se a métrica *Lines of Code* (LOC). Segundo Peters (2001, p. 433) “LOC é a forma mais dominante na classe de medidas de *software* orientadas ao tamanho”. Com o auxílio dessa métrica analisou-se as linhas de código do sistema, o que resultou no quadro abaixo:

**Quadro 3. Métrica de Linha de Código**

<b>SISTEMA</b>		
<b>MÉTRICAS</b>	<b>SEM COMPONENTE</b>	<b>COM COMPONENTE</b>
<b>LOC</b> Número de Linhas de Código	3.517	2.986

## **2. Quanto a Coesão**

O sistema Manutenção foi desenvolvido utilizando os conceitos da orientação a objeto que pregam naturalmente a modularização. No sistema, cada método implementado realiza somente uma função, assim como no componente.

Ao utilizar o componente no sistema nenhuma regra de negócio necessita ser alterada, pois é o componente que se adapta para atender estas regras. Com isto esta parte do sistema se torna independente do restante executando uma atividade específica, resultando em um alto nível de coesão.

## **3. Quanto ao acoplamento**

O acoplamento do sistema Manutenção ao Componente LJ é de nível baixo. Isso pode ser comprovado pela quantidade mínima de parâmetros para sua utilização. A quantidade de parâmetros foi reduzida ao máximo para que se por ventura ocorrer alguma alteração, a mesma afete o mínimo possível o sistema.

## **4. Quanto ao tempo de desenvolvimento**

Durante a implementação do sistema e do componente foi medido, para ambos, o tempo de seu desenvolvimento. A codificação do sistema com as consultas internas ocorreram em um curto espaço de tempo se comparado ao componente que levou três vezes mais. Essa diferença se justifica pelo fato de que para implementar o componente foi necessário primeiramente estudar as tecnologias envolvidas, para só então iniciar seu desenvolvimento.

A partir disso, sugere-se a necessidade de realizar uma análise aprofundada - tempo, pessoas capacitadas e tecnologias disponíveis - antes de desenvolver componentes. É importante também considerar se o componente será reutilizado em outras partes do sistema e/ou em outros projetos para não desperdiçar os recursos que seriam empregados no seu desenvolvimento.

### 5. Quanto ao tamanho da aplicação

Chegou-se a conclusão que os ganhos ao utilizar um componente com uma funcionalidade específica será sempre de 1: n. Onde teremos **1** (um) componente para **n** partes de um sistema e/ou projetos, independente do seu tamanho.

No caso do sistema e do componente desenvolvidos fez-se uso da métrica *Vocabulary Size* (VS), que segundo Figueiredo (2005) é para avaliar o tamanho do sistema em termos de número de classes ou interfaces. Os resultados obtidos são demonstrados no Quadro 4.

**Quadro 4. Métrica de Tamanho do Vocabulário**

<b>SISTEMA</b>		
<b>MÉTRICAS</b>	<b>SEM COMPONENTE</b>	<b>COM COMPONENTE</b>
<b>VS</b> Tamanho do Vocabulário	38 classes ou 12 interfaces	34 classes ou 9 interfaces

### 6. Quanto a reutilização

A reutilização é muito grande se o sistema utilizar o componente para realizar as consultas. No entanto, se essa funcionalidade for implementada dentro da aplicação, a mesma irá se repetir infinitas vezes interferindo no reuso de código, já que copiar e colar não é reutilizar.

Assim, como recomendação de uso, propõe-se utilizar as técnicas abordadas neste trabalho, pois como foi mostrado em toda a validação há um ganho considerável. No entanto, é importante lembrar as pessoas, que vão iniciar seus projetos utilizando essas práticas, que o tempo de desenvolvimento aumenta e que é necessário investimentos em mão-de-obra, treinamento e tecnologias. Com certeza os resultados vão compensar esse esforço inicial e o produto final será de qualidade. O tempo de desenvolvimento será reduzido com os módulos, componentes e até *frameworks* reutilizáveis que estarão disponíveis para serem usados a qualquer momento.

## 6 CONCLUSÃO

A aplicação da ciência da computação foi fundamental para o desenvolvimento do trabalho. Os objetivos traçados foram alcançados e os resultados estão descritos no capítulo de validação. No entanto, alguns itens referentes à arquitetura de *software* tiveram o seu desenvolvimento prejudicado, pois durante a realização do curso não houve disciplinas que abordassem o assunto com maior profundidade. Além disso, os livros abordam esse assunto de forma totalmente teórica, sem exemplos práticos de como é modelada e implementada. Isso dificultou a visualização através de modelos, de uma arquitetura que aplica as técnicas de modularização e componentização. O resultado foi um modelo de arquitetura de *software* sem um modelo de referencia, mas que demonstrou o objetivo do trabalho.

Já o processo de desenvolvimento RUP utilizado em todo o desenvolvimento do sistema e do componente foi muito importante, pois permitiu o acompanhamento de todas as fases do desenvolvimento e facilitou o entendimento de algumas atividades que não são comuns no dia-a-dia. Aliado a isso, a utilização das técnicas de modularização e componentização mostraram novas maneiras de analisar e desenvolver sistemas com o objetivo de reutilizar a maior quantidade de código possível. Além disso, observou-se a importância de manter alto grau de coesão e baixo acoplamento em relação a componentes,

módulos e frameworks reutilizáveis para que os sistemas, que venham a utilizá-los, tenham o mínimo de alterações.

Em relação aos outros assuntos abordados, a bibliografia compõe-se de autores que demonstram claramente como a modularização e componentização são importantes, descrevendo como e onde essas práticas devem ser utilizadas para que se consiga bons resultados.

Durante o desenvolvimento do sistema e do componente tivemos dificuldades por não dominarmos algumas tecnologias que foram utilizadas. No entanto, com o decorrer da implementação essas dúvidas foram sanadas, principalmente em relação à codificação do componente, pois tínhamos um conhecimento básico de XML.

Fazer este trabalho foi muito importante para nosso crescimento pessoal e profissional, pois permitiu conhecer e empregar conceitos e técnicas pouco utilizadas no desenvolvimento de sistema de *software*. A aplicação de uma metodologia de desenvolvimento mostrou o quão fundamental são as atividades que antecedem a implementação do produto de *software*, já que no dia-a-dia essa prática é pouco utilizada. Para nós foi um grande desafio, mas com muito trabalho e esforço chegamos ao nosso objetivo. Vemos isso como uma vitória pessoal e principalmente profissional, pois é no mercado de trabalho que isso realmente vai fazer diferença.

Dessa forma, para as pessoas que tiverem o interesse de conhecer mais sobre as técnicas de modularização e componentização dentro da arquitetura de *software*, sugerimos que estudem autores como STAA (2000), PRESSMAN (1995), MENDES (2002), além de outros que abordam conteúdos a respeito de engenharia de *software* e arquitetura de *software*. Em relação à arquitetura de *software*, é difícil de encontrar livros que abordam esse assunto com profundidade, mas os livros que existem dão suporte para iniciar nessa prática.

Pretendemos para trabalhos futuros, primeiramente utilizar a monografia para publicar um artigo sobre a utilização da modularização e componentização num projeto de *software*. Além disso, fazer uma pós-graduação em uma das áreas abordadas neste projeto, para aprofundar os conhecimentos adquiridos.

Contudo, a monografia permitiu, para nós orientandos, ter uma nova visão sobre como desenvolver sistemas, valorizar a reutilização durante os processo de desenvolvimento para reduzir os custos e evitar que a manutenção se torne um problema por causa da redundância de código-fonte.



## REFERÊNCIAS

ALLEN, Paul; BAMBARA, Joseph. **Sun Certified Enterprise Architect for J2EE**: guia oficial de certificação. Rio de Janeiro: Campus, 2003. 613 p.

ANT. **Apache Ant**. The Apache Ant Project. Disponível em: <<http://www.ant.apache.org>>. Acesso em: 20 ago. 2006

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**. Rio de Janeiro: Campus, 2002. 286 p.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: Guia do Usuário. Rio de Janeiro: Campus, 2000. 472 p.

DENNIS, Alan; WIXON, Barbara Haley. **Análise e Projeto de Sistemas**. 2 ed. Rio de Janeiro: LTC, 2005. 484 p.

ECLIPSE. **Eclipse IDE**. Eclipse Foundation Inc. Disponível em: <<http://www.eclipse.org>>. Acesso em: 20 ago. 2006

FIGUEIREDO, Eduardo Magno Lages; STAA, Arndt von. Avaliação de um Modelo de Qualidade para Implementações Orientadas a Objetos e Orientadas a Aspectos. **Pontifícia Universidade Católica do Rio de Janeiro**. Rio de Janeiro, 2005. n.14/05. 29 p.

HIBERNATE. **Relacional Persistence for Java and .NET**. Disponível em: <<http://www.hibernate.org>>. Acesso em: 20 ago. 2006.

HUSTED, Ted et al. **Struts em Ação**. Rio de Janeiro: Editora Ciência Moderna Ltda, 2004. 604 p.

JAVA. **Sun Developer Network**. Disponível em: <<http://www.java.sun.com>>. Acesso em: 20 ago. 2006.

LARMAN, Craig. **Utilizando UML e padrões**: Uma introdução à análise e ao projeto orientado a objetos. Porto Alegre: Bookman, 2000. 491 p.

LEITE, J. C. **Engenharia de Requisitos**: Notas de Aula. Rio de Janeiro: PUC-RJ, 1994.

MENDES, Antonio. **Arquitetura de Software**: desenvolvimento orientado para arquitetura. Rio de Janeiro: Campus, 2002. 212 p.

MICROSOFT. **Microsoft Internet Explorer**. Disponível em: <<http://www.microsoft.com/brasil/windows/ie/default.mspx>>. Acesso em: 20 ago. 2006.

PAGE-JONES, Meilir. **Fundamentos do Desenho Orientado a Objeto com UML**. São Paulo: Makron Books, 2001. 462 p.

PETERS, James F.; PEDRYCZ, Witold. **Engenharia de Software**: teoria e prática. Rio de Janeiro: Campus, 2001. 602 p.

PRESSMAN, Roger S. **Engenharia de Software**. 1 ed. São Paulo: Makron Books, 1995. 1056 p.

RATIONAL. **IBM Rational Unified Process – RUP**. Sistema de Metodologia de Desenvolvimento. Estados Unidos da América: Rational, 2006. Disponível em: <http://www.ibm.com/br/products/software/rational/rup.phtml>.

SANCHES, Rosely. **Subprogramas, funções e procedimentos**. Desenvolvido pela Universidade Federal da Bahia. Disponível em: <<http://twiki.im.ufba.br/pub/MAT146/ToDoMaterial/slides-aula-funcoes.pdf>>. Acesso em: 09 mar. 2006.

SILVA, Edna Lúcia da; MENEZES, Estera Muszkat. **Metodologia da pesquisa e elaboração de dissertação**. 4 ed. Florianópolis: UFSC, 2005.

SOMMERVILLE, Ian. **Engenharia de Software**. 6 ed. São Paulo: Addison Wesley, 2003. 592 p.

STAA, Arndt von. **Programação Modular**: desenvolvendo programas complexos de forma organizada e segura. Rio de Janeiro: Campus, 2000. 690 p.

STRUTS. **Struts Framework**. The Apache Software Foundation. Disponível em: <<http://struts.apache.org/>>. Acesso em: 20 ago. 2006.

TOMCAT. **Apache Tomcat**. Container Web. The Apache Software Foundation. Disponível em: <<http://www.tomcat.apache.org>>. Acesso em: 20 ago. 2006

TONSIG, Sérgio Luiz. **Engenharia de Software**: Análise e Projeto de Sistemas. São Paulo: Futura, 2003. 351 p.

VAROTO. Ane Cristina. **Visões em Arquitetura de Software**. 2002. 108 f. Dissertação (Mestrado em Ciência da Computação) – Universidade de São Paulo, São Paulo. Disponível em: <[www.ime.usp.br/dcc/posgrad/teses/ane.pdf](http://www.ime.usp.br/dcc/posgrad/teses/ane.pdf)>. Acesso em: 10 abr. 2006.

XML. **Extensible Markup Language**. Disponível em: <<http://www.w3.org/XML>>. Acesso em: 20 ago. 2006.

## **APÊNDICES**

## **APÊNDICE A – Artefato do RUP de Plano de Desenvolvimento de *Software***

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Software Development Plan (Small Project)**  
**Plano de Desenvolvimento de Software (Pequenos Projetos)**

**Versão 1.2**

<b>Disciplina:</b>	<i>Project Management</i> - Gerência do Projeto
<b>Papel:</b>	<i>Project Manager</i> - Gerente do Projeto
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.2
Software Developent Plan (Small Project)	Data de Criação: 30-mai-2006
Arquivo: A – Plano de Desenvolvimento de Software (Pequenos Projetos).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
30/05/2006	1.0	Processo inicial do projeto	Luciane e Jefferson
20/08/2006	1.1	Revisão da fase de Elaboração	Luciane Pires Werlang
05/11/2006	1.2	Revisão final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.2
Software Development Plan (Small Project)	Data de Criação: 30-mai-2006
Arquivo: A – Plano de Desenvolvimento de Software (Pequenos Projetos).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Escopo	4
1.3	Definições, Acrônimos e Abreviações	4
1.4	Visão Geral	4
2.	Visão Geral do Projeto	4
2.1	Propósito do Projeto, Escopo e Objetivos	4
2.2	Suposições e Restrições	5
2.3	Objetos De Entrega do Projeto	5
2.4	Evolução do Software Development Plan (Plano de Desenvolvimento de Software)	5
3.	Organização do Projeto	5
3.1	Estrutura Organizacional	5
3.2	<i>Roles</i> (Papéis) e Responsabilidades	5
4.	Processo de Gerenciamento	5
4.1	Estimativas do Projeto	5
4.2	Plano de Projeto	5
4.2.1	Planejamento da Fase	5
4.2.2	Objetivos das Iterações	6
4.2.3	Liberações	6
4.2.4	Cronograma do Projeto	6
4.2.5	Recursos do Projeto	6
4.3	Monitoramento de Projeto e Controle	6
5.	Anexos	6



Sistema Manutenção	Versão: 1.2
Software Development Plan (Small Project)	Data de Criação: 30-mai-2006
Arquivo: A – Plano de Desenvolvimento de Software (Pequenos Projetos).doc	

# Software Development Plan (Small Project)

## Plano de Desenvolvimento de Software (Pequenos Projetos)

### 1. Introdução

O Plano de Desenvolvimento de Software é um documento completo, composto de artefatos que reúnem as informações necessárias para gerenciar um projeto.

#### 1.1 Objetivo

O objetivo do *Software Development Plan* (Plano de Desenvolvimento de Software) é reunir as informações necessárias para controlar o projeto. Além disso, é um plano de alto nível que descreve a abordagem utilizada para o desenvolvimento do software.

As pessoas descritas a seguir usam o *Software Development Plan* (Plano de Desenvolvimento de Software):

- O **project manager (gerente de projeto)** usa este documento para construir o planejamento do projeto e os recursos necessários, e para seguir o planejamento sem atraso.
- **Project team members (Membros da equipe de Projeto)** usam o mesmo para entender o que eles precisam fazer, quando eles precisam fazer isso, e outras atividades que estão sob sua responsabilidade.

#### 1.2 Escopo

Este *Software Development Plan* (Plano de Desenvolvimento de Software) descreve todo o plano a ser usado pelo Projeto *Component Search*. Os detalhes das iterações individuais serão descritos nos *Iteration Plans* (Planos de Iteração).

Os requisitos do sistema serão descritos no *Software Requirements Specification* (Especificação de Requisitos de Software). Por outro lado, os detalhes de Arquitetura serão descritos no *Software Architecture Document* (Documento de Arquitetura de Software).

Como o projeto é focado nos componentes, existirão documentos específicos para descrevê-los.

#### 1.3 Definições, Acrônimos e Abreviações

Estão descritas no Glossário (APÊNDICE J).

#### 1.4 Visão Geral

Este *Software Development Plan* (Plano de Desenvolvimento de Software) contém as seguintes informações:

Visão Geral do Projeto — fornece uma descrição do propósito do projeto, escopo, e objetivos. Este também define quando as entregas do projeto são esperadas para ocorrer.

Organização de Projeto — descreve a estrutura organizacional da equipe do projeto.

Processo de Gerenciamento — explica o custo estimado e o planejamento, define as principais fases e os *milestones* (marcos) para o projeto, e descreve como o projeto será monitorado.

### 2. Visão Geral do Projeto

#### 2.1 Propósito do Projeto, Escopo e Objetivos

Desenvolver um sistema de controle de clientes por estado e cidade, sendo que as consultas serão realizadas da maneira tradicional e também através de um componente. Esse componente será implementado em linguagem de desenvolvimento JAVA e XML para uso genérico, tendo como principal característica o reuso de software.

Sistema Manutenção	Versão: 1.2
Software Development Plan (Small Project)	Data de Criação: 30-mai-2006
Arquivo: A – Plano de Desenvolvimento de Software (Pequenos Projetos).doc	

## 2.2 Suposições e Restrições

O desenvolvimento do sistema tem como linguagem de desenvolvimento JAVA, baseado na arquitetura MVC e Struts.

O componente é desenvolvido em JAVA e XML, sua ligação com o protótipo é feito através do uso de TagLib.

A equipe de desenvolvimento é composta somente por dois integrantes.

Não teremos orçamento externo para desenvolver o sistema, pois o orçamento para o seu desenvolvimento é pequeno.

## 2.3 Objetos De Entrega do Projeto

Os itens entregues serão CD contendo o código fonte, o arquivo executável e a documentação do sistema e do componente.

## 2.4 Evolução do Software Development Plan (Plano de Desenvolvimento de Software)

O *Software Development Plan* (Plano de Desenvolvimento de Software) é revisado antes do início de cada *Iteration Phase* (Fase de Iteração).

## 3. Organização do Projeto

Neste item são descritas a estrutura organizacional do projeto com os seus papéis e responsabilidades.

### 3.1 Estrutura Organizacional

Analista e Desenvolvedor: fazem toda a parte de análise de sistema, bem como o levantamento dos requisitos e o desenvolvimento do sistema.

### 3.2 Roles (Papéis) e Responsabilidades

Pessoa	Rational Unified Process Role (Papel)
Luciane Pires Werlang	Gerente de Projeto Analista de Sistemas Arquiteto de Software Especificador de Requisitos Projetista Programador
Jefferson Amorim de Oliveira	Gerente de Projeto Analista de Sistemas Arquiteto de Software Especificador de Requisitos Projetista Programador

## 4. Processo de Gerenciamento

A seguir serão listados alguns itens do Processo de Gerenciamento.

### 4.1 Estimativas do Projeto

Este projeto não tem custo de desenvolvimento, nem receberá auxílio financeiro.

### 4.2 Plano de Projeto

Esta seção contém os planejamentos e os recursos para o projeto.

#### 4.2.1 Planejamento da Fase

Abaixo está descrito as principais metas para cada fase.

- Conceção: Analisar o problema, definir o sistema, definir o escopo do sistema, levantar os requisitos para iniciar a modelagem do software para que este permita o reuso.
- Elaboração: Definir a arquitetura que atenda os requisitos do sistema, que sirva de base para a

Sistema Manutenção	Versão: 1.2
Software Development Plan (Small Project)	Data de Criação: 30-mai-2006
Arquivo: A – Plano de Desenvolvimento de Software (Pequenos Projetos).doc	

análise de consistência e dependência, além de prover suporte ao reuso. Dar continuidade a análise dos requisitos e a finalização da modelagem.

- Construção: Realizar a codificação de software e os testes.
- Transição: Realizar a finalização do software, para que seja feita a distribuição.

#### 4.2.2 *Objetivos das Iterações*

Concretizar cada iteração, fazendo as devidas correções e integrando-as de forma eficiente com as próximas, para a finalização do projeto.

#### 4.2.3 *Liberações*

O sistema será distribuído no final do projeto, porém antes não haverá uma versão beta para que seja disponibilizado para a realização de testes e satisfação de clientes.

#### 4.2.4 *Cronograma do Projeto*

Os cronogramas estão todos demonstrados nos Planos de Iteração de cada fase, ou seja, Plano de Iteração Concepção (APÊNDICE F), Plano de Iteração Elaboração (APÊNDICE F), Plano de Iteração Construção (APÊNDICE F) e Plano de Iteração Transição (APÊNDICE F).

#### 4.2.5 *Recursos do Projeto*

As pessoas envolvidas no projeto deverão ter conhecimento em Banco de Dados, domínio em programação JAVA, conhecimentos em linguagem XML.

### 4.3 **Monitoramento de Projeto e Controle**

Segue uma lista de itens a considerar:

- *Requirements Management* (Gerenciamento de Requisitos)

Os requisitos para este sistema são capturados no documento *Software Requirements Specification* (Especificação de Requisitos de Software).

- *Software Architecture Document* (Documento de Arquitetura de Software)

Neste documento está descrita a arquitetura do sistema.

## 5. **Anexos**

O projeto seguirá o processo do RUP for Small Projects (RUP para Projetos Pequenos), customizado para que seja adaptável a este projeto.

## **APÊNDICE B – Artefato do RUP de Especificação Complementar**

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Supplementary Specification**  
**Especificação Complementar**

**Versão 1.1**

<b>Disciplina:</b>	<i>Requirements</i> - Requisitos
<b>Papel:</b>	<i>System Analyst</i> - Analista de Sistema
<b>Indivíduo:</b>	Luciane Werlang & Jefferson Oliveira

Sistema Manutenção	Versão: 1.1
Supplementary Specification	Data de Criação: 28-mai-2006
Arquivo: B – Especificação Complementar.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
30/05/2006	1.0	Edição Inicial	Jefferson Oliveira
20/08/2006	1.1	Revisão e atualização	Luciane Pires Werlang

Sistema Manutenção	Versão: 1.1
Supplementary Specification	Data de Criação: 28-mai-2006
Arquivo: B – Especificação Complementar.doc	

## Sumário

1.	Introdução	4
1.1	Definições, Acrônimos e Abreviações.	4
2.	Funcionalidade	4
3.	Usabilidade	4
3.1	Interfaces	4
4.	Confiabilidade	4
4.1	Disponibilidade	4
5.	Suportabilidade	4
5.1	Manutenibilidade	4
5.2	Portabilidade	4
6.	Restrição de Design	4
6.1	Tecnologia	4
6.2	Ferramentas	5
6.3	Banco de Dados	5
6.4	Arquitetura	5
7.	Componentes Adquiridos	5
8.	Interfaces	5
8.1	Interfaces de Usuário	5
8.2	Interfaces de Hardware	5
8.3	Interfaces de Software	5
8.4	Interfaces de Comunicação	5
9.	Solicitação de Licença	5
10.	Padrões Aplicáveis	5

Sistema Manutenção	Versão: 1.1
Supplementary Specification	Data de Criação: 28-mai-2006
Arquivo: B – Especificação Complementar.doc	

# Supplementary Specification

## Especificação Complementar

### 1. Introdução

A **Supplementary Specification (Especificação Complementar)** descreve os requisitos do sistema que não são capturados de imediato nos modelos de caso de uso. Tais requisitos incluem:

- Requisitos legais e reguladores, incluindo padrões de aplicações.
- Atributos de qualidade do sistema para serem construídos, incluindo: usabilidade, confiabilidade, performance e requisitos suportáveis.
- Outros requisitos assim como sistemas operacionais e ambientes, requisitos de compatibilidade e restrições de projeto.

#### 1.1 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

### 2. Funcionalidade

Os requisitos funcionais serão capturados via especificação de casos de uso.

### 3. Usabilidade

Esta seção descreve os requisitos que afetam a usabilidade do sistema.

#### 3.1 Interfaces

As Interfaces serão amigáveis e suportadas pelos browsers mais conhecidos (Internet Explorer, Firefox).

### 4. Confiabilidade

Esta seção descreve os requisitos que afetam a confiabilidade do sistema.

#### 4.1 Disponibilidade

O sistema está disponível em uma plataforma WEB, podendo ser acessado a qualquer momento, desde que o usuário esteja previamente cadastrado.

### 5. Suportabilidade

Esta seção descreve os requisitos que afetam a suportabilidade do sistema.

#### 5.1 Manutenibilidade

O sistema é projetado e implementado utilizando os conceitos de modularização. Isso facilitará e tornará a reparação de erros ou defeitos mais rápida, assim como a adição de novas funcionalidades ou modificações. A arquitetura utilizada para construir o sistema visa a independência das camadas de implementação em Modelo, Visão e Controle além de oferecer a vantagem da modularidade. Permite também que partes já desenvolvidas em um sistema possam ser reutilizadas em novos sistemas.

#### 5.2 Portabilidade

Como o sistema é desenvolvido em Java, ele poderá ser executado em diferentes plataformas, desde que estas possuem a respectiva JVM (*Java Virtual Machine*).

### 6. Restrição de Design

Esta seção descreve as restrições de design que afetam o sistema.

#### 6.1 Tecnologia

A linguagem de modelagem que é adotada para projetar o sistema é a UML (*Unified Modeling Language*), para codificação do sistema é utilizada a linguagem de programação Java e XML com o



Sistema Manutenção	Versão: 1.1
Supplementary Specification	Data de Criação: 28-mai-2006
Arquivo: B – Especificação Complementar.doc	

auxílio do *Web Container* Apache Tomcat. Para criação das interfaces é utilizada as tecnologias HTML, JSP, XLS, DTD, JavaScript e Servlets.

## 6.2 Ferramentas

O desenho da modelagem será realizada na ferramenta *Enterprise Architect* (EA). A descrição dos artefatos serão feitos no *Microsoft Word*. A codificação do sistema será realizada no Eclipse.

## 6.3 Banco de Dados

O Banco de Dados que dará suporte a aplicação será o Oracle.

## 6.4 Arquitetura

A arquitetura utilizada será a arquitetura MVC, pelo suporte que ela dá a modularidade e reuso de código.

## 7. Componentes Adquiridos

Os componentes adquiridos para auxiliar na implementação do sistema serão os *Frameworks Jakarta Struts e Hibernate*.

## 8. Interfaces

Esta seção descreve os requisitos que afetam as interfaces do sistema.

### 8.1 Interfaces de Usuário

As interfaces do sistema serão de fácil utilização buscando utilizar recursos amigáveis e de rápido carregamento pelo browser.

### 8.2 Interfaces de Hardware

Não haverá necessidade de nenhuma interface de hardware.

### 8.3 Interfaces de Software

Não haverá necessidade de nenhuma interface de software.

### 8.4 Interfaces de Comunicação

O sistema irá rodar em cima dos protocolos de comunicação HTTP e TCP/IP, tanto em redes locais como fora da organização.

## 9. Solicitação de Licença

Será necessário adquirir licença para o Banco de Dados Oracle, assim como para o sistema operacional Windows, caso utilize-o.

## 10. Padrões Aplicáveis

Não se aplica.

## **APÊNDICE C – Artefato do RUP de Especificação de Requisitos de *Software***

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção  
Software Requirements Specification (with Use-  
Case)**

**Especificação de Requisitos de Software**

**Versão 1.2**

<b>Disciplina:</b>	<i>Requirements</i> - Requisitos
<b>Papel:</b>	<i>Requirements Specifier</i> - Especificador de Requisitos
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.2
Software Requirements Specification (with Use-Case)	Data de Criação: 28-mai-2006
Arquivo: C – Especificação de Requisitos de Software.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
30/05/2006	1.0	Edição Inicial	Jefferson Oliveira
20/08/2006	1.1	Revisão	Luciane Pires Werlang
05/11/2006	1.2	Revisão Final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.2
Software Requirements Specification (with Use-Case)	Data de Criação: 28-mai-2006
Arquivo: C – Especificação de Requisitos de Software.doc	

## Sumário

1.	Introdução	4
1.1	Definições, Acrônimos e Abreviações.	4
2.	Descrição Completa	4
2.1	Análise do Modelo de Caso de Uso	4
2.1.1	Atores	4
2.1.2	Casos de Uso	4
3.	Requisitos Específicos	7
3.1	Requisitos Funcionais	7
3.2	Requisitos Não-Funcionais	7
3.3	Requisitos Complementares	7

Sistema Manutenção	Versão: 1.2
Software Requirements Specification (with Use-Case)	Data de Criação: 28-mai-2006
Arquivo: C – Especificação de Requisitos de Software.doc	

# Software Requirements Specification (with Use-Case)

## Especificação de Requisitos de Software

### 1. Introdução

O *Software Requirements Specification* – SRS (Especificação dos Requisitos do Software) captura os requisitos de software por completo para o sistema, ou para uma parte do sistema. Seguindo um típico esboço do SRS para um projeto que usa modelagem de caso de uso. Este artefato consiste em um pacote contendo casos de uso do modelo de casos de uso, especificação suplementar adequada e outras informações de apoio. Para um modelo de um SRS que não use modelagem de casos de uso, o qual captura todos os requisitos em um simples documento, com as seções adequadas inseridas da Especificação Complementar.

#### 1.1 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

### 2. Descrição Completa

Esta seção do *Software Requirements Specification* - SRS (Especificação de Requisitos de Software) descreve as causas gerais que afetam o produto e seus requisitos. Esta seção não determina requisitos específicos. No lugar disto ele fornece um fundo para aqueles requisitos, os quais são definidos em detalhes na Seção 3, e torna-os fácil de compreender.

#### 2.1 Análise do Modelo de Caso de Uso

Contém uma visão geral do modelo de caso de uso e atores, junto com os diagramas adequados e relacionamentos.

##### 2.1.1 Atores

Listagem de todos os Atores que irão utilizar o sistema.

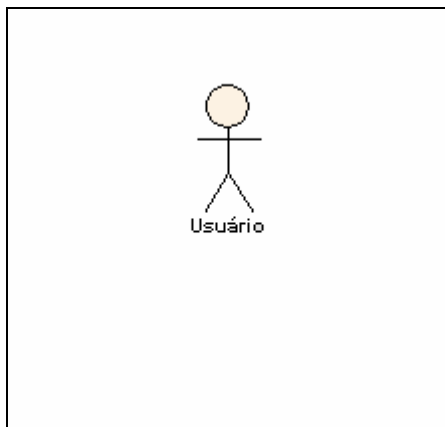


Figura 1. Atores

- Usuário: Representa todos os usuários do sistema.

##### 2.1.2 Casos de Uso

Listagem de todos os Casos de Uso do sistema.

Sistema Manutenção	Versão: 1.2
Software Requirements Specification (with Use-Case)	Data de Criação: 28-mai-2006
Arquivo: C – Especificação de Requisitos de Software.doc	

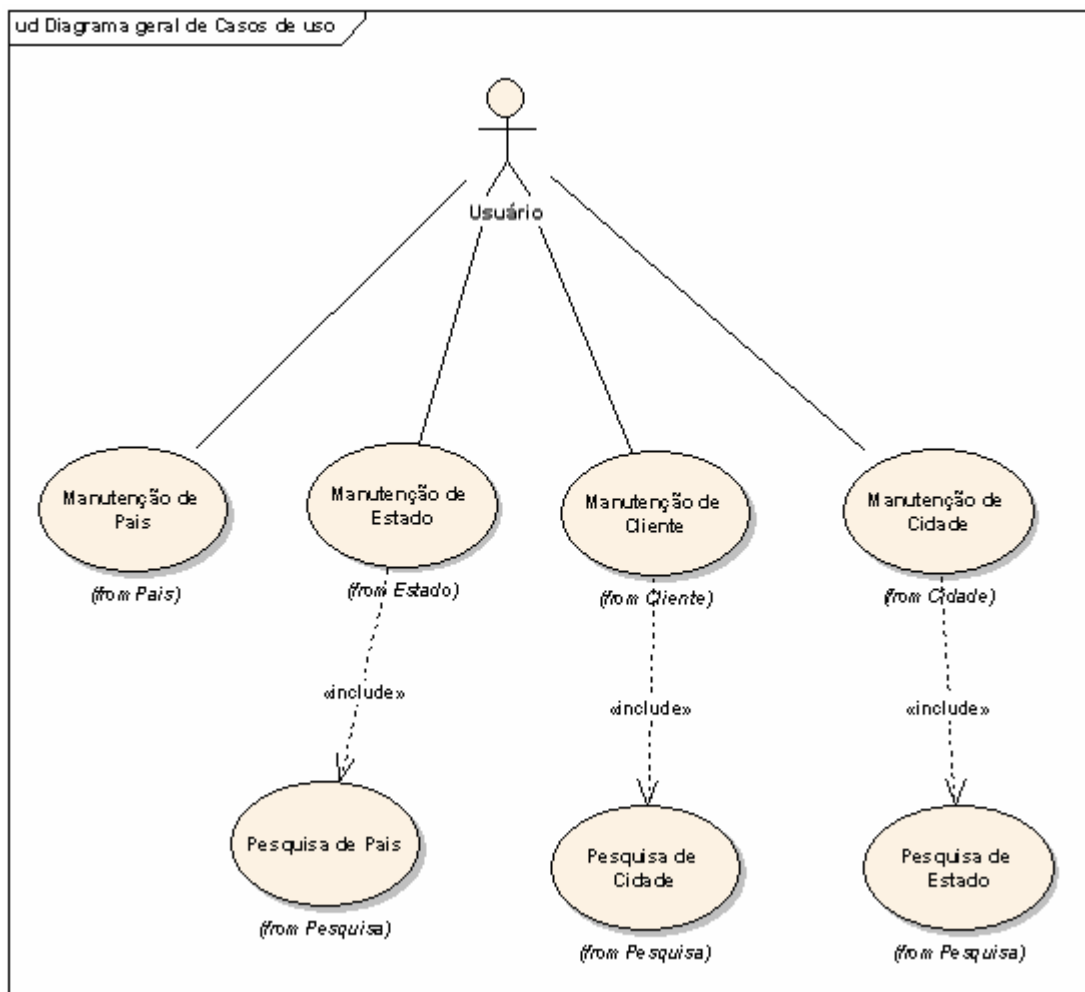


Figura 2. Casos de Uso

O diagrama de atividade a seguir demonstra uma visão geral do funcionamento do sistema proposto.

Sistema Manutenção	Versão: 1.2
Software Requirements Specification (with Use-Case)	Data de Criação: 28-mai-2006
Arquivo: C – Especificação de Requisitos de Software.doc	

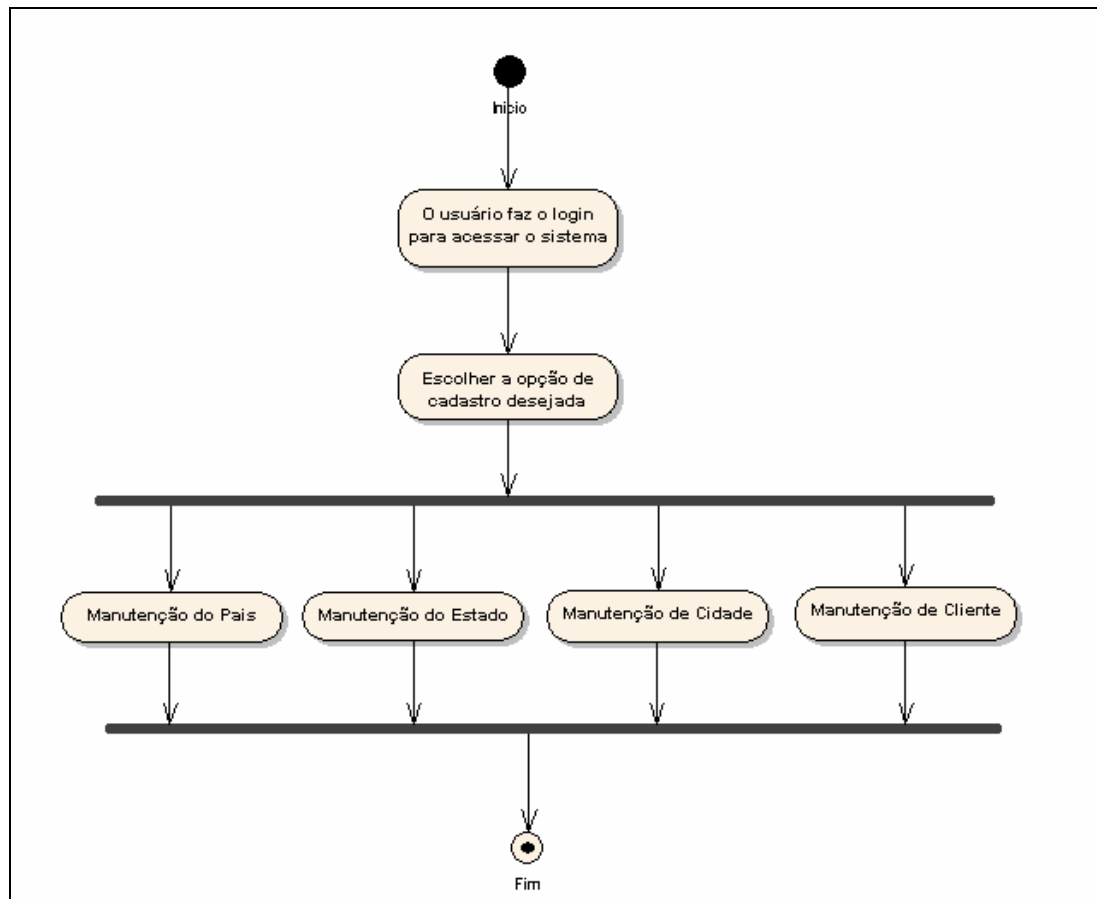


Figura 3. Diagrama de atividade dos casos de uso

Para entender melhor os casos de uso, eles foram divididos em pacotes.

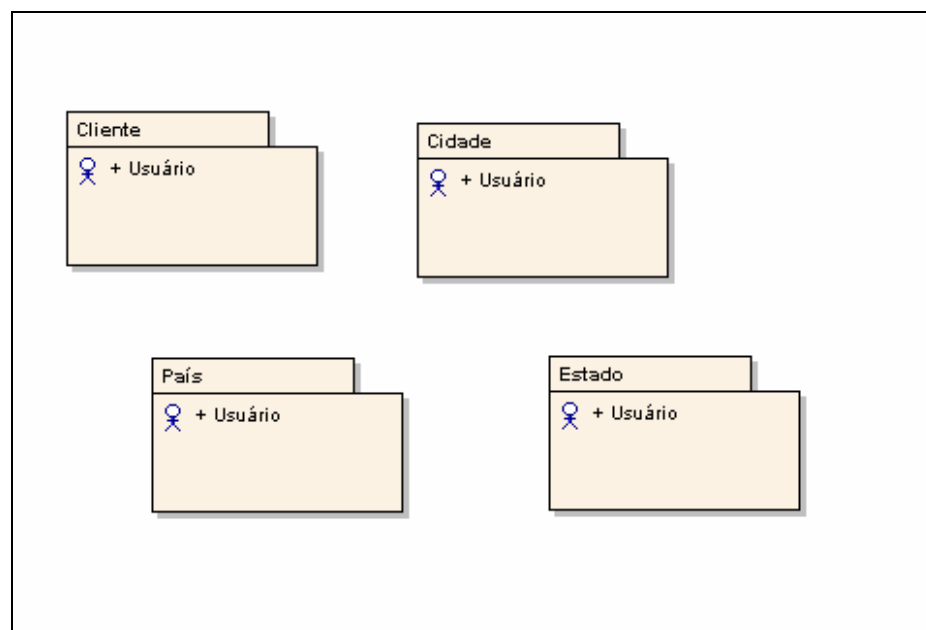


Figura 4. Pacotes que compõem o sistema



Sistema Manutenção	Versão: 1.2
Software Requirements Specification (with Use-Case)	Data de Criação: 28-mai-2006
Arquivo: C – Especificação de Requisitos de Software.doc	

### 3. Requisitos Específicos

Esta seção do **Software Requirements Specification (Especificação dos Requisitos de Software)** contém todos os requisitos de software para um nível de detalhes suficiente para habilitar os designers a fazer o design do sistema. Eles estão divididos em requisitos funcionais e requisitos não-funcionais.

#### 3.1 Requisitos Funcionais

Em todos os requisitos funcionais citados a seguir o usuário realizará as operações de cadastro, atualização, exclusão e consulta. Para o usuário ter acesso às operações de atualização e exclusão, ele deverá primeiramente executar a operação de consulta para então entrar na edição do item selecionado. Para os requisitos de manutenção de Estado, manutenção de Cidade e manutenção de Cliente o usuário utilizará um componente que executará consultas genéricas que irão preencher campos que pertencem a outro cadastro.

- Manutenção de País
- Manutenção de Estado
- Manutenção de Cidade
- Manutenção de Cliente

#### 3.2 Requisitos Não-Funcionais

Para acessar o sistema o usuário deverá estar cadastrado na Base de Dados com login e senha. Essa restrição se dá pelo uso de um sistema de segurança, que evita que pessoas não autorizadas possam acessar o sistema e corromper seus dados.

#### 3.3 Requisitos Complementares

Estão descritos no Artefato Especificação Complementar (APÊNDICE B)

## **APÊNDICE D – Artefato do RUP de Especificação de Caso de Uso**

---

**Luciane Werlang & Jefferson de Oliveira**

---

**Sistema Manutenção**  
**Use-Case Specifications**  
**Especificação dos Casos de Uso**

**Versão 1.2**

<b>Disciplina:</b>	<i>Requirements</i> - Requisitos
<b>Papel:</b>	<i>Requirements Specifier</i> - Especificador de Requisitos
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
31/05/2006	1.0	Edição Inicial	Jefferson Oliveira
20/08/2006	1.1	Revisão	Luciane e Jefferson
05/11/2006	1.2	Revisão Final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

## Sumário

1.0	Manutenção de Cidade	4
1.1	Descrição Resumida	4
1.2	Fluxo de Eventos	4
1.3	Fluxo Básico	4
1.4	Fluxo Alternativo	4
1.4.1	Incluir	4
1.4.2	Excluir	5
1.4.3	Consultar	5
1.4.4	Atualizar	5
2.0	Manutenção de Cliente	5
2.1	Descrição Resumida	6
2.2	Fluxo de Eventos	6
2.3	Fluxo Básico	6
2.4	Fluxo Alternativo	6
2.4.1	Incluir	6
2.4.2	Excluir	6
2.4.3	Consultar	6
2.4.4	Atualizar	6
3.0	Manutenção de País	6
3.1	Descrição Resumida	7
3.2	Fluxo de Eventos	7
3.3	Fluxo Básico	7
3.4	Fluxo Alternativo	7
3.4.1	Incluir	7
3.4.2	Excluir	7
3.4.3	Consultar	8
3.4.4	Atualizar	8
4.0	Manutenção de Estado	8
4.1	Descrição Resumida	8
4.2	Fluxo de Eventos	9
4.3	Fluxo Básico	9
4.4	Fluxo Alternativo	9
4.4.1	Incluir	9
4.4.2	Excluir	9
4.4.3	Consultar	9
4.4.4	Atualizar	9
5.0	Pré-Condições	9
5.1	Conexão com o Banco de Dados	9
5.2	Usuário cadastrado	9
6.0	Pós-Condições	9
6.1	Executar a ação	9

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

# Use-Case Specifications

## Especificação dos Casos de Uso

### 1.-1 Manutenção de Cidade

O diagrama de atividade descreve de um modo geral como será o funcionamento da manutenção de cidade.

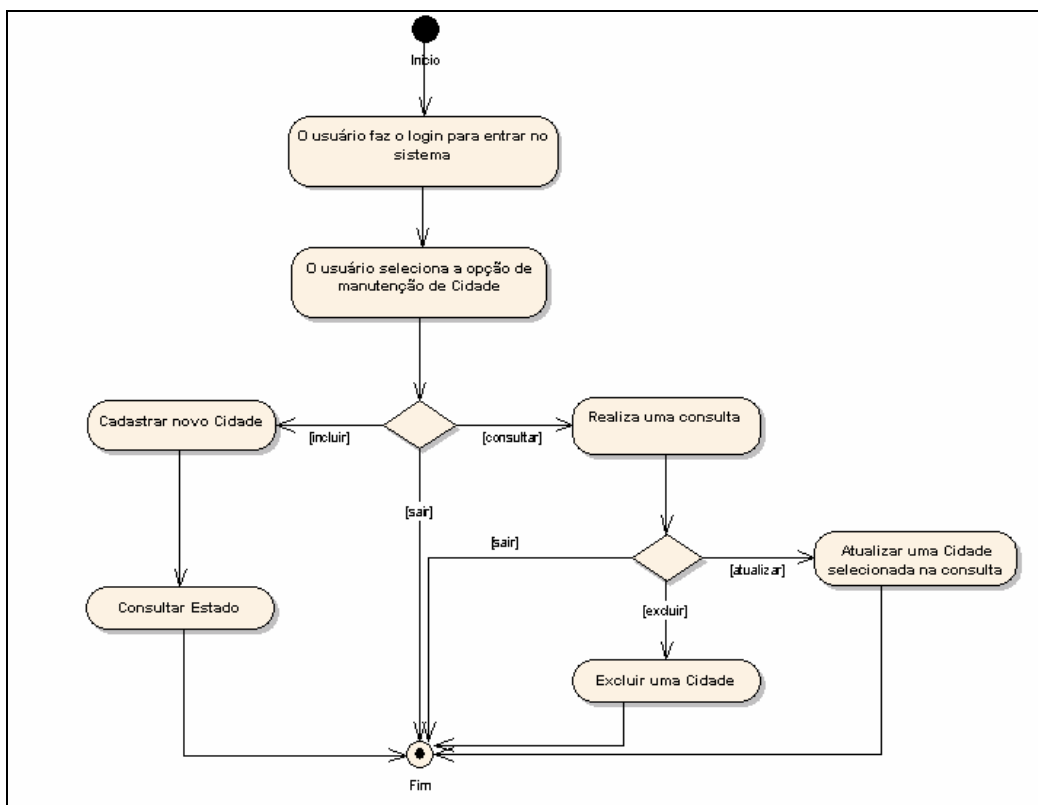


Figura 1. Diagrama de atividade do pacote Cidade

#### 1.1 Descrição Resumida

O usuário realizará a manutenção de Cidade no sistema. As operações que serão realizadas são: Incluir, Excluir, Atualizar e Consultar.

#### 1.2 Fluxo de Eventos

#### 1.3 Fluxo Básico

- 1- O usuário clica na opção Cidade.
- 2- O usuário visualizará uma tela onde ele poderá consultar ou fazer a inclusão de uma nova cidade.
- 3- Escolhendo a opção de consultar, o usuário poderá excluir ou alterar a cidade selecionada.
- 4- Caso seja nova inclusão, abrirá para o usuário uma nova tela, onde ele fará o cadastro.

#### 1.4 Fluxo Alternativo

##### 1.4.1 Incluir

- 1- O sistema mostrará um formulário com os campos disponíveis para preenchimento.
- 2- O usuário preenche os campos.

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

3- O usuário invoca o Caso de Uso Consultar.

4- Usuário finaliza sua inclusão.

#### 1.4.2 Excluir

1- O usuário executa o Consultar.

2- O usuário seleciona a(s) linha(s) referente(s) a(s) cidade(s) a ser(em) apagada(s).

3- Usuário confirma a opção de apagar a(s) cidade(s).

4- Caso ocorra algum erro será emitida uma mensagem.

#### 1.4.3 Consultar

1- O usuário poderá utilizar filtro para realizar a consulta.

2- O usuário realiza a consulta.

3- Sistema mostrará uma listagem.

#### 1.4.4 Atualizar

1- O usuário executa o Consultar.

2- O usuário seleciona a linha referente a cidade a ser atualizada.

3- Sistema mostrará o formulário com os campos preenchidos e habilitados para serem alterados.

4- Usuário altera o(s) campo(s) e salva as informações.

## 2.-1 Manutenção de Cliente

O diagrama de atividade descreve de um modo geral como será o funcionamento da manutenção de cliente.

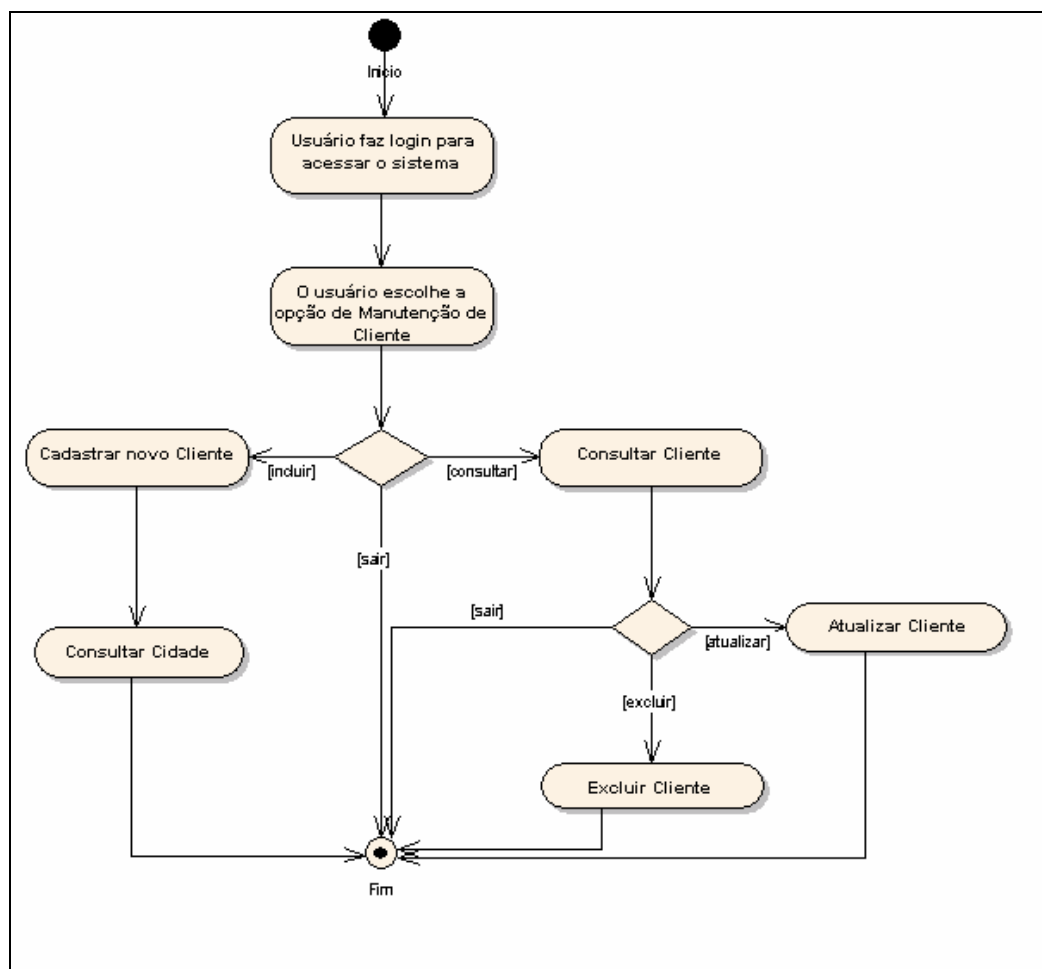


Figura 2. Diagrama de atividade do pacote Cliente

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

## 2.1 Descrição Resumida

O usuário realizará a manutenção de Cliente no sistema. As operações que serão realizadas são: Incluir, Excluir, Atualizar e Consultar.

## 2.2 Fluxo de Eventos

### 2.3 Fluxo Básico

- 1- O usuário clica na opção Cliente.
- 2- O usuário visualizará uma tela onde ele poderá consultar ou fazer a inclusão de um novo cliente.
- 3- Escolhendo a opção de consultar, o usuário poderá excluir ou alterar o cliente selecionado.
- 4- Caso seja nova inclusão, abrirá para o usuário uma nova tela, onde ele fará o cadastro.

### 2.4 Fluxo Alternativo

#### 2.4.1 Incluir

- 1- O sistema mostrará um formulário com os campos disponíveis para preenchimento.
- 2- O usuário preenche os campos.
- 3- O usuário invoca o Caso de Uso Consultar.
- 4- Usuário finaliza sua inclusão.

#### 2.4.2 Excluir

- 1- O usuário executa o Consultar.
- 2- O usuário seleciona a(s) linha(s) referente(s) ao(s) cliente(s) a ser(em) apagado(s).
- 3- Usuário confirma a opção de apagar o(s) cliente(s).
- 4- Caso ocorra algum erro será emitida uma mensagem.

#### 2.4.3 Consultar

- 1- Usuário poderá utilizar filtro para realizar a consulta.
- 2- O usuário realiza a consulta.
- 3- Sistema mostrará uma listagem.

#### 2.4.4 Atualizar

- 1- O usuário executa o Consultar.
- 2- O usuário seleciona a linha referente ao cliente a ser atualizado.
- 3- Sistema mostrará o formulário com os campos preenchidos e habilitados para serem alterados.
- 4- Usuário altera o(s) campo(s) e salva as informações.

## 3.-1 Manutenção de País

O diagrama de atividade descreve de um modo geral como será o funcionamento da manutenção de país.



Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

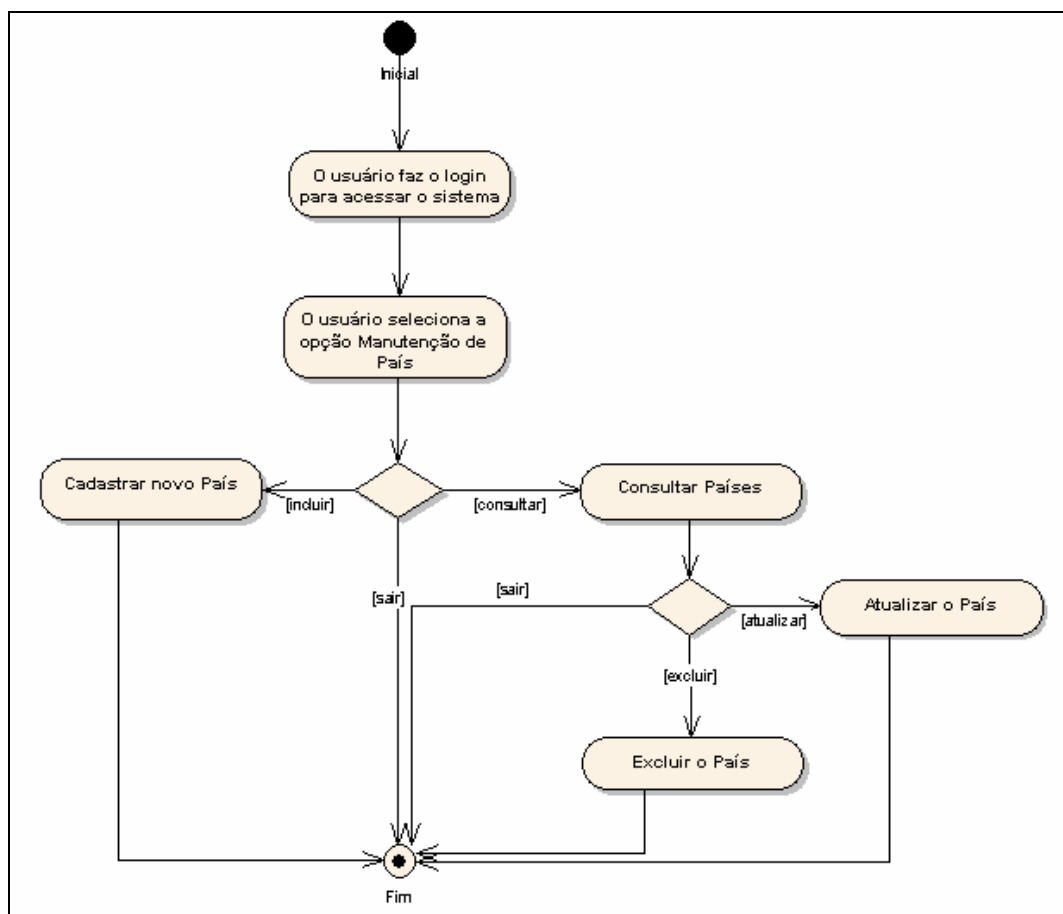


Figura 3. Diagrama de atividade do pacote País

### 3.1 Descrição Resumida

O usuário realizará a manutenção de País no sistema. As operações que serão realizadas são: Incluir, Excluir, Atualizar e Consultar.

### 3.2 Fluxo de Eventos

### 3.3 Fluxo Básico

- 1- O usuário clica na opção País.
- 2- O usuário visualizará uma tela onde ele poderá consultar ou fazer a inclusão de um novo país.
- 3- Escolhendo a opção de consultar, o usuário poderá excluir ou alterar o país selecionado.
- 4- Caso seja nova inclusão, abrirá para o usuário uma nova tela, onde ele fará o cadastro.

### 3.4 Fluxo Alternativo

#### 3.4.1 Incluir

- 1- O sistema mostrará um formulário com os campos disponíveis para preenchimento.
- 2- O usuário preenche os campos.
- 3- Usuário finaliza sua inclusão.

#### 3.4.2 Excluir

- 1- O usuário executa o Consultar.
- 2- O usuário seleciona a(s) linha(s) referente(s) ao(s) país(es) a ser(em) apagado(s).
- 3- Usuário confirma a opção de apagar o(s) país(es).
- 4- Caso ocorra algum erro será emitida uma mensagem.

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

### 3.4.3 Consultar

- 1- O usuário poderá utilizar filtro para realizar a consulta.
- 2- O usuário realiza a consulta.
- 3- Sistema mostrará uma listagem.

### 3.4.4 Atualizar

- 1- O usuário executa o Consultar.
- 2- O usuário seleciona a linha referente ao país a ser atualizado.
- 3- Sistema mostrará o formulário com os campos preenchidos e habilitados para serem alterados.
- 4- Usuário altera o(s) campo(s) e salva as informações.

## 4.0 Manutenção de Estado

O diagrama de atividade descreve de um modo geral como será o funcionamento da manutenção de estado.

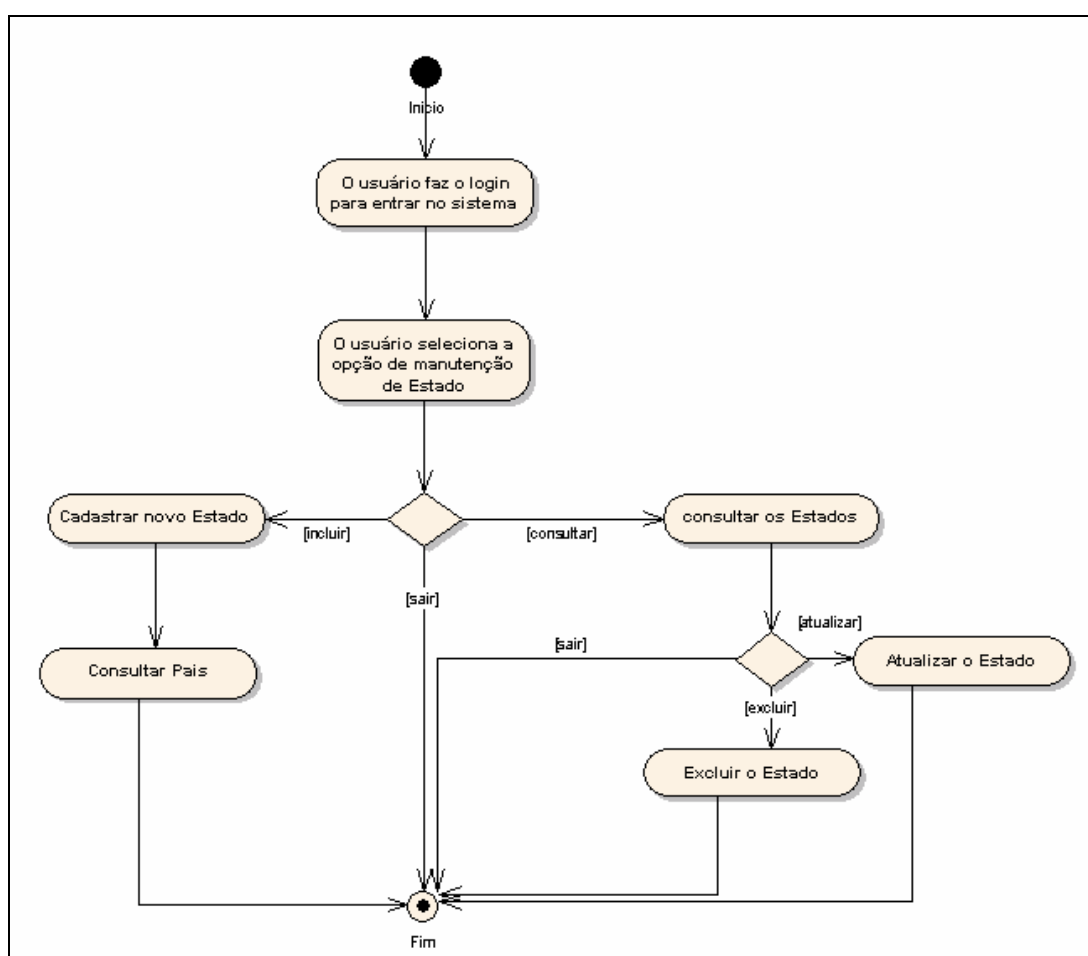


Figura 4. Diagrama de atividade do pacote Estado

### 4.1 Descrição Resumida

O usuário realizará a manutenção de Estado no sistema. As operações que serão realizadas são: Incluir, Excluir, Atualizar e Consultar.

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 7-jun-2006
Arquivo: D - Especificação dos Casos de Uso .doc	

## 4.2 Fluxo de Eventos

### 4.3 Fluxo Básico

- 1- O usuário clica na opção Estado.
- 2- O usuário visualizará uma tela onde ele poderá consultar ou fazer a inclusão de um novo estado.
- 3- Escolhendo a opção de consultar, o usuário poderá excluir ou alterar o estado selecionado.
- 4- Caso seja nova inclusão, abrirá para o usuário uma nova tela, onde ele fará o cadastro.

### 4.4 Fluxo Alternativo

#### 4.4.1 Incluir

- 1- O sistema mostrará um formulário com os campos disponíveis para preenchimento.
- 2- O usuário preenche os campos.
- 3- O usuário invoca o Caso de Uso Consultar.
- 4- Usuário finaliza sua inclusão.

#### 4.4.2 Excluir

- 1- O usuário executa o Consultar.
- 2- O usuário seleciona a(s) linha(s) referente(s) ao(s) estado(s) a ser(em) apagado(s).
- 3- Usuário confirma a opção de apagar o(s) estado(s).
- 4- Caso ocorra algum erro será emitida uma mensagem.

#### 4.4.3 Consultar

- 1- O usuário poderá utilizar filtro para realizar a consulta.
- 2- O usuário realiza a consulta.
- 3- Sistema mostrará uma listagem.

#### 4.4.4 Atualizar

- 1- O usuário executa o Consultar.
- 2- O usuário seleciona a linha referente ao estado a ser atualizado.
- 3- Sistema mostrará o formulário com os campos preenchidos e habilitados para serem alterados.
- 4- Usuário altera o(s) campo(s) e salva as informações.

## 5.-1 Pré-Condições

Uma pré-condição de um caso de uso é o estado que o sistema deve apresentar antes do caso de uso ser executado.

### 5.1 Conexão com o Banco de Dados

A conexão com o banco deve estar ativa.

### 5.2 Usuário cadastrado

O usuário deve estar devidamente cadastrado para utilizar o sistema.

## 6.-1 Pós-Condições

Uma pós-condição de um caso de uso é uma lista dos possíveis estados que o sistema pode estar imediatamente após a finalização de um caso de uso.

### 6.1 Executar a ação

Mostrar o resultado da ação.

---

**Luciane Werlang & Jefferson de Oliveira**

---

**Sistema Manutenção**  
**Use-Case Specification**  
**Especificação do Caso de Uso Pesquisar**  
**Versão 1.2**

<b>Disciplina:</b>	<i>Requirements</i> - Requisitos
<b>Papel:</b>	<i>Requirements Specifier</i> - Especificador de Requisitos
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 28-mai-2006
Arquivo: D - Especificação do Caso de Uso Pesquisar.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
31/05/2006	1.0	Edição Inicial	Jefferson Oliveira
20/08/2006	1.1	Revisão	Luciane e Jefferson
05/11/2006	1.2	Revisão final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 28-mai-2006
Arquivo: D - Especificação do Caso de Uso Pesquisar.doc	

## Sumário

1.	Pesquisar	4
1.1	Descrição Resumida	5
2.	Fluxo de Eventos	5
2.1	Fluxo Básico	5
3.	Pré-Condições	5
3.1	Conexão com o Banco de Dados	5
3.2	Usuário cadastrado	5
3.3	Dados cadastrados	5
4.	Pós-Condições	5
4.1	Executar a ação	5

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 28-mai-2006
Arquivo: D - Especificação do Caso de Uso Pesquisar.doc	

# Use-Case Specification

## Especificação do Caso de Uso Pesquisar

### 1. Pesquisar

O Diagrama de Atividade representa o funcionamento do caso de uso.

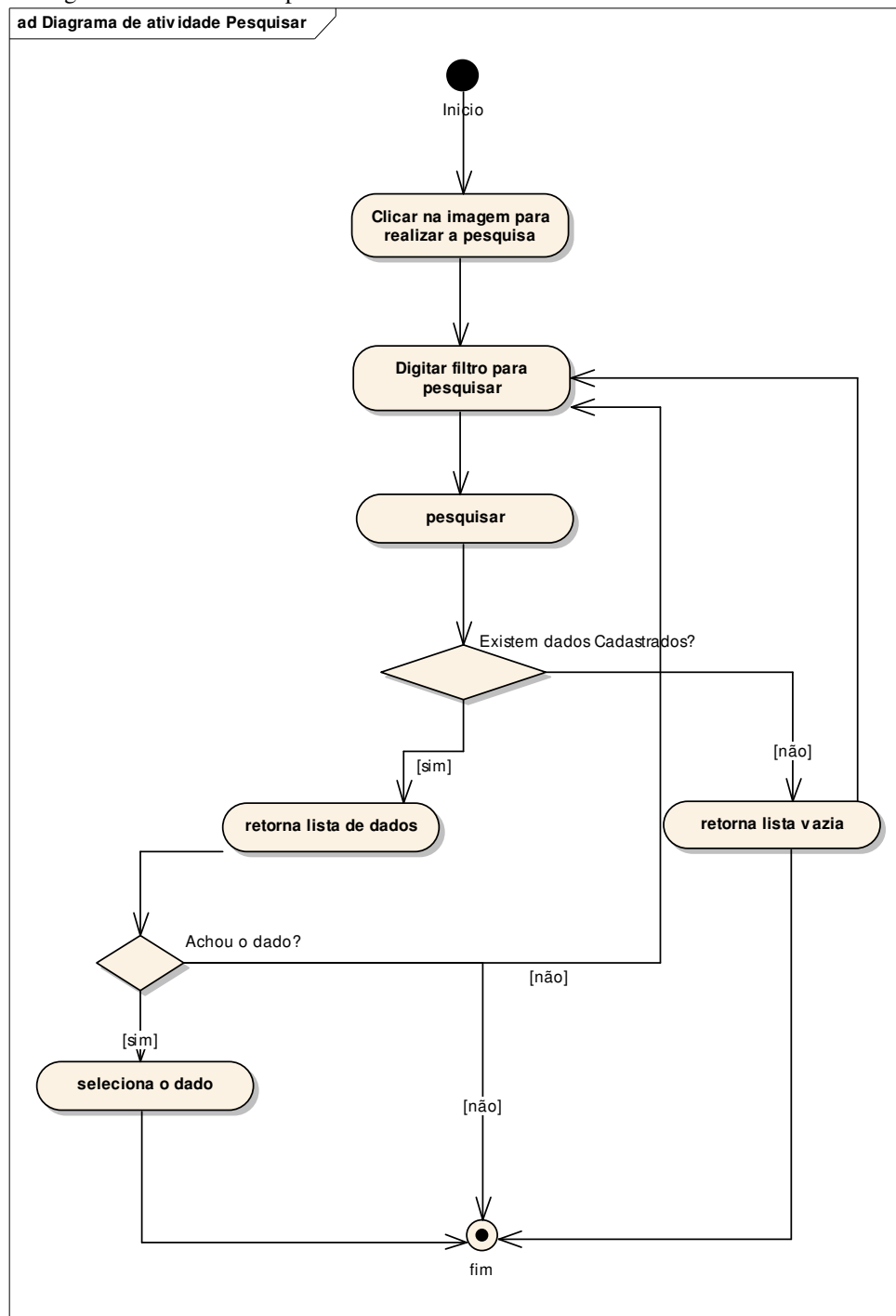


Figura 1. Diagrama de Atividade do Caso de Uso Pesquisar

Sistema Manutenção	Versão: 1.2
Use-Case Specification	Data de Criação: 28-mai-2006
Arquivo: D - Especificação do Caso de Uso Pesquisar.doc	

### 1.1 Descrição Resumida

O usuário realiza a pesquisa para preencher um campo que necessita desse dado.

## 2. Fluxo de Eventos

### 2.1 Fluxo Básico

- 1- O usuário clica numa imagem de procura que abrirá uma tela.
- 2- A tela terá filtros chaves para realizar a consulta do dado a ser inserido no cadastro.
- 3- A consulta retornará uma lista.
- 4- O usuário poderá selecionar somente um item da lista para ser inserido no cadastro.
- 5- Após a seleção, ele deverá clicar em selecionar para que o item seja enviado para o cadastro e a tela se fechará.

## 3. Pré-Condições

Uma pré-condição de um caso de uso é o estado que o sistema deve apresentar antes do caso de uso ser executado.

### 3.1 Conexão com o Banco de Dados

A conexão com o banco deve estar ativa.

### 3.2 Usuário cadastrado

O usuário deve estar devidamente cadastrado para utilizar o sistema.

### 3.3 Dados cadastrados

Os dados devem ser cadastrados para que tenha algum retorno.

## 4. Pós-Condições

O dado selecionado deve estar retornado nos campos para o qual foi requisitado

### 4.1 Executar a ação

Mostrar o resultado da ação.



## **APÊNDICE E – Artefato do RUP de Documento de Arquitetura de *Software***

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Software Architecture Document**  
**Documento de Arquitetura de Software**

**Versão 1.2**

<b>Disciplina:</b>	<i>Analysis &amp; Design</i> - Análise e Design
<b>Papel:</b>	<i>Software Architect</i> - Arquiteto de Software
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
30/05/2006	1.0	Edição Inicial	Jefferson Oliveira
08/10/2006	1.1	Revisão	Luciane Werlang
05/11/2006	1.2	Revisão final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
2.	Representação Arquitetural	4
3.	Metas Arquiteturais e Restrições	4
4.	Realização do Caso de Uso	5
5.	Visão Lógica	5
5.1	Diagrama Geral de Visão	6
5.2	Diagrama Geral de Controle	7
5.3	Diagrama Geral de Modelo	7
6.	Visão de Implementação	8

Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

# Software Architecture Document

## Documento de Arquitetura de Software

### 1. Introdução

A introdução do *Software Architecture Document (Documento de Arquitetura de Software)* fornece uma visão geral do conteúdo do documento. Ele inclui o objetivo, escopo, definições, acrônimos, abreviações, referências e a visão geral deste documento.

#### 1.1 Objetivo

Este documento fornece uma abrangente visão geral da arquitetura do sistema, usando um número de visões arquiteturais diferentes para descrever diferentes aspectos do sistema. Pretende-se capturar e conduzir as decisões da arquitetura significativas que foram feitas no sistema.

#### 1.2 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J)

### 2. Representação Arquitetural

Conforme previsto no artefato Especificação Complementar (APÊNDICE B) o sistema será desenvolvido e implementado segundo uma arquitetura em três camadas, visando interdependência das camadas para facilitar a manutenção e sua construção. Durante o design do sistema, o mapeamento das camadas respeitou a divisão dos pacotes detectados.

### 3. Metas Arquiteturais e Restrições

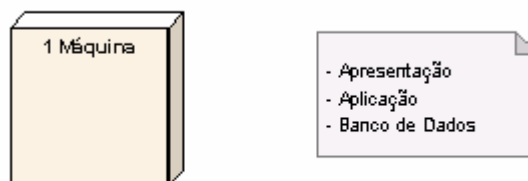
O sistema em desenvolvimento possui as seguintes características:

- **Multiplataforma:** o sistema baseado em tecnologia Java, que é independente de plataforma para implementação.
- **Banco de Dados:** Como o sistema possui interdependência entre as camadas, o banco de dados se comunica junto à camada de persistência sem comprometer o restante do sistema. Ou seja, o sistema será portátil para qualquer banco padrão SQL-92 acessado via JDBC.
- **Camadas Lógicas:** o sistema é projetado na arquitetura MVC:
  - Visão (View): Arquivos JSP e Java (do tipo Form – Bean) que são carregados pelo cliente via browser, fazendo a ligação entre o usuário e a aplicação.
  - Controle (Controller): determina o fluxo da apresentação servindo como uma camada intermediária entre a camada de apresentação e a lógica.
  - Modelo (Model): É o coração da aplicação. Responsável por tudo que a aplicação vai fazer. Modela os dados e o comportamento por trás do processos de negócio, preocupa-se apenas com o armazenamento, manipulação e geração de dados, além de ser um encapsulamento de dados e de comportamento independente da Visão.

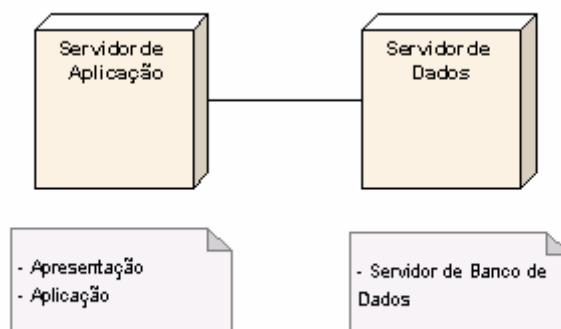
O sistema no contexto físico é compatível com as seguintes arquiteturas:

- 1 Máquina: sistema acessado via browser do próprio servidor onde esta hospedado contendo toda a aplicação (visão, controle e modelo) juntamente com o banco de dados.

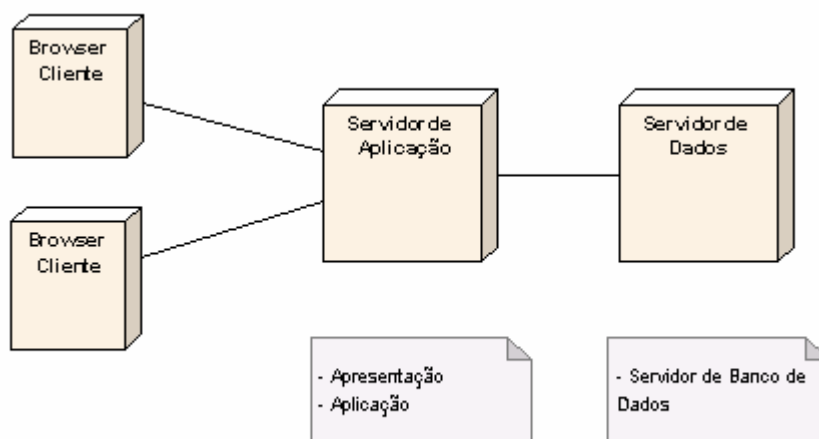
Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	



- 2 Máquinas: sistema acessado via browser do próprio servidor onde esta hospedado contendo a aplicação e um servidor para os dados.



- 3 Máquinas: interface via browser na máquina dos clientes, um servidor de aplicação para a apresentação e controle e um servidor para os dados.



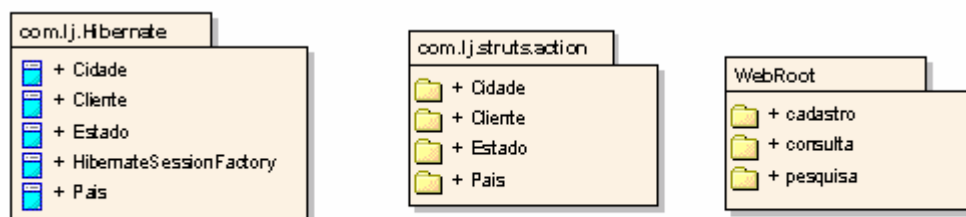
#### 4. Realização do Caso de Uso

Todas as realizações dos casos de uso foram feitas com base na arquitetura MVC: Modelo, Visão e Controle.

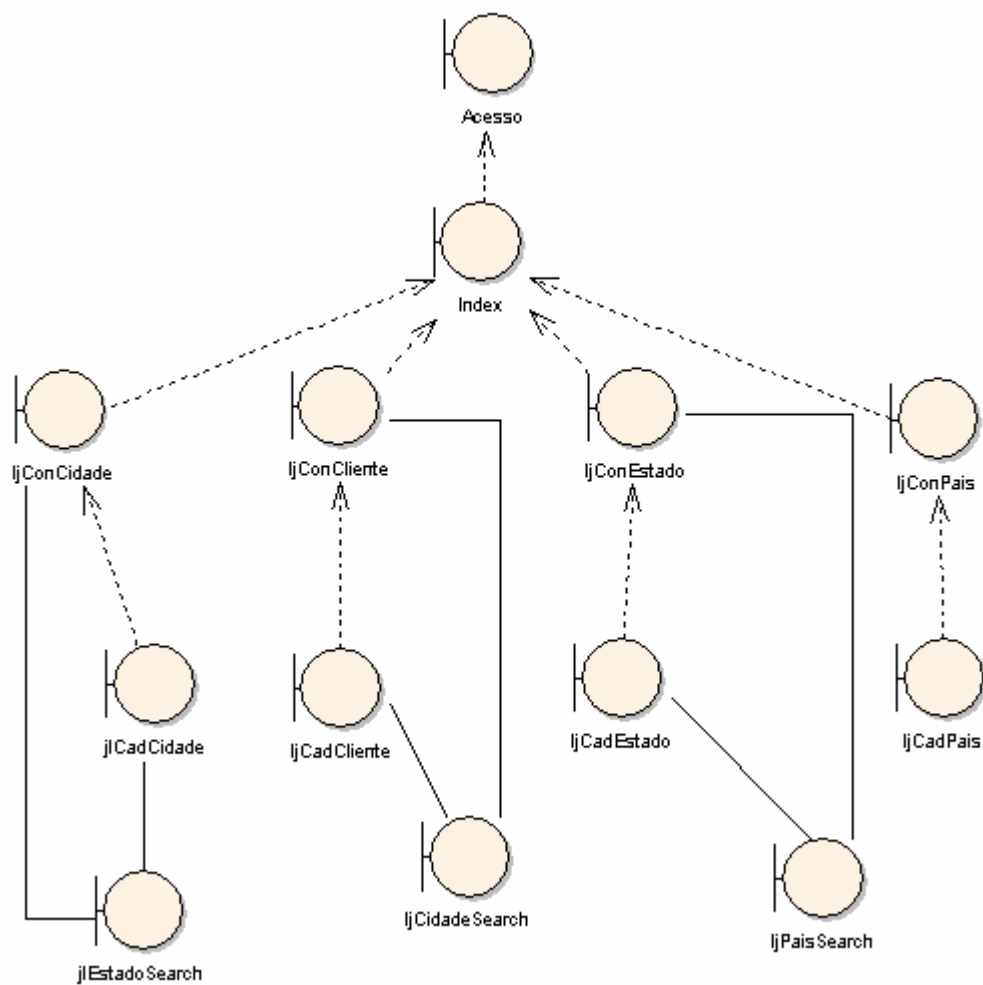
#### 5. Visão Lógica

Esta seção descreve as partes significativas da arquitetura do modelo de design, tais como sua composição em subsistemas e em pacotes.

Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

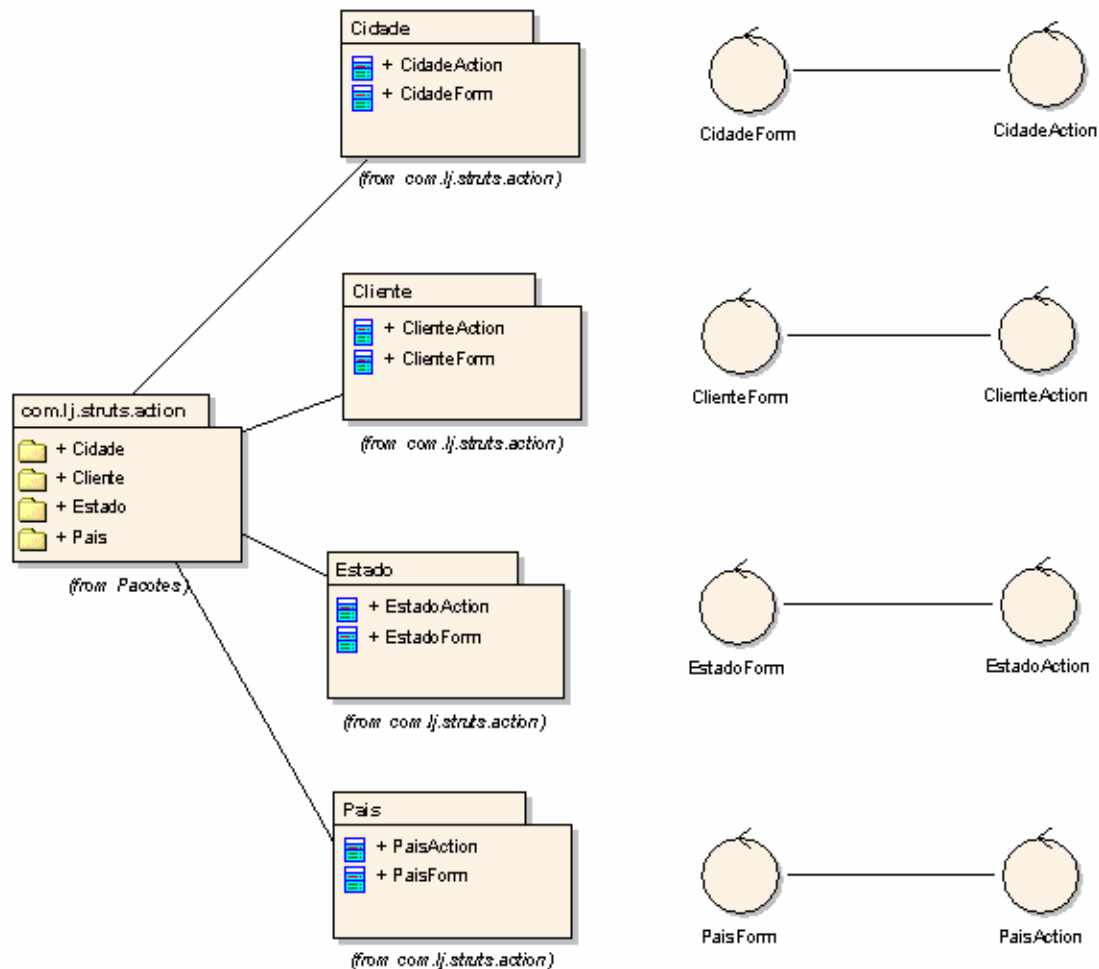


## 5.1 Diagrama Geral de Visão

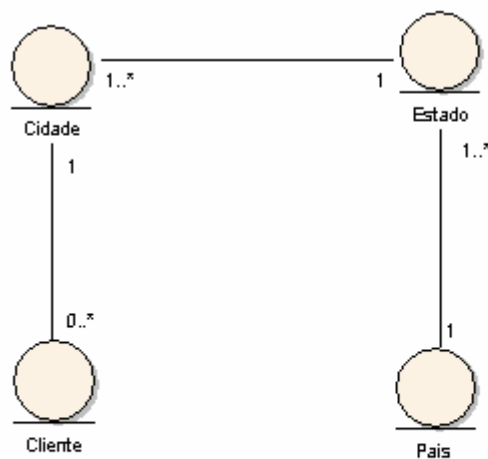


Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

## 5.2 Diagrama Geral de Controle



## 5.3 Diagrama Geral de Modelo

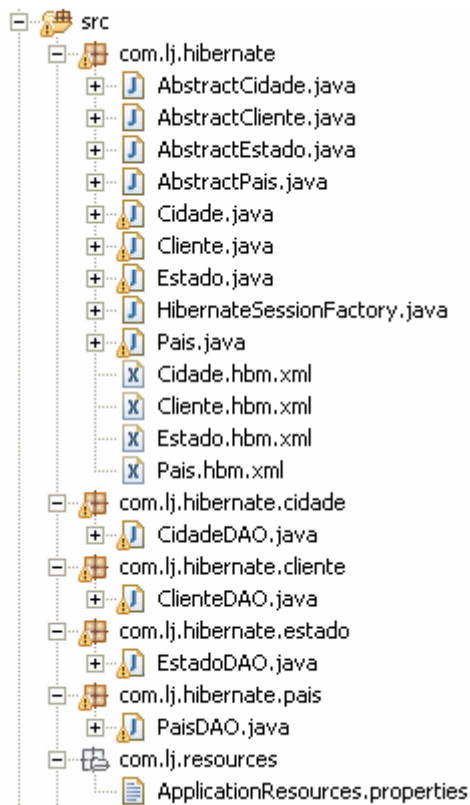




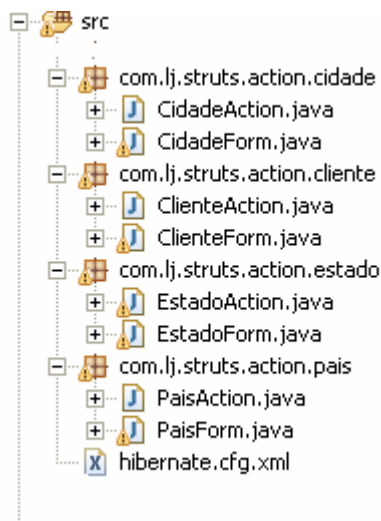
Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

## 6. Visão de Implementação

### ▪ Camada de Modelo

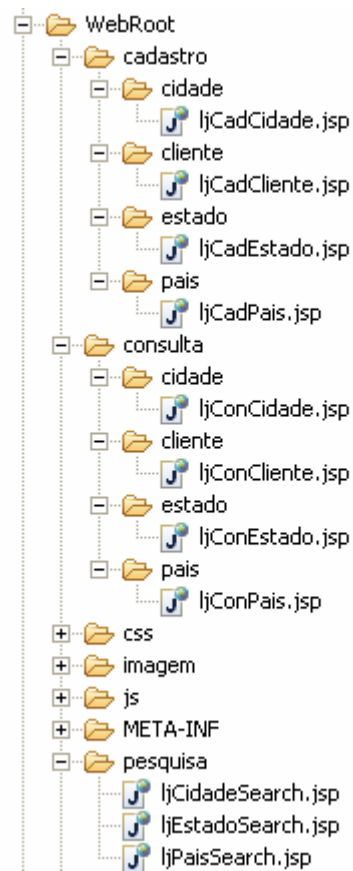


### ▪ Camada de Controle



Sistema Manutenção	Versão: 1.2
Software Architecture Document	Data de Criação: 28-mai-2006
Arquivo: E - Documento de Arquitetura de Software.doc	

▪ **Camada de Visão**



## **APÊNDICE F – Artefato do RUP de Plano de Iteração**

---

**Luciane Werlang & Jefferson de Oliveira**

---

**Sistema Manutenção  
Iteration Plan  
Plano de Iteração (Concepção)**  
**Versão 1.1**

<b>Disciplina:</b>	<i>Project Management</i> - Gerência de Projeto
<b>Papel:</b>	<i>Project Manager</i> - Gerente de Projeto
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Concepção).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
13/06/2006	1.0	Edição Inicial	Luciane e Jefferson
05/11/2006	1.1	Revisão Final	Luciane Werlang

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Concepção).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
1.3	Visão Geral	4
2.	Cronograma	4

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Concepção).doc	

# Iteration Plan

## Plano de Iteração (Concepção)

### 1. Introdução

O documento propõe demonstrar um plano para a fase de concepção do projeto.

#### 1.1 Objetivo

O objetivo é mostrar um cronograma contendo todas as atividades associadas a essa fase, juntamente com as datas e seus respectivos responsáveis.

#### 1.2 Definições, Acrônimos e Abreviações.

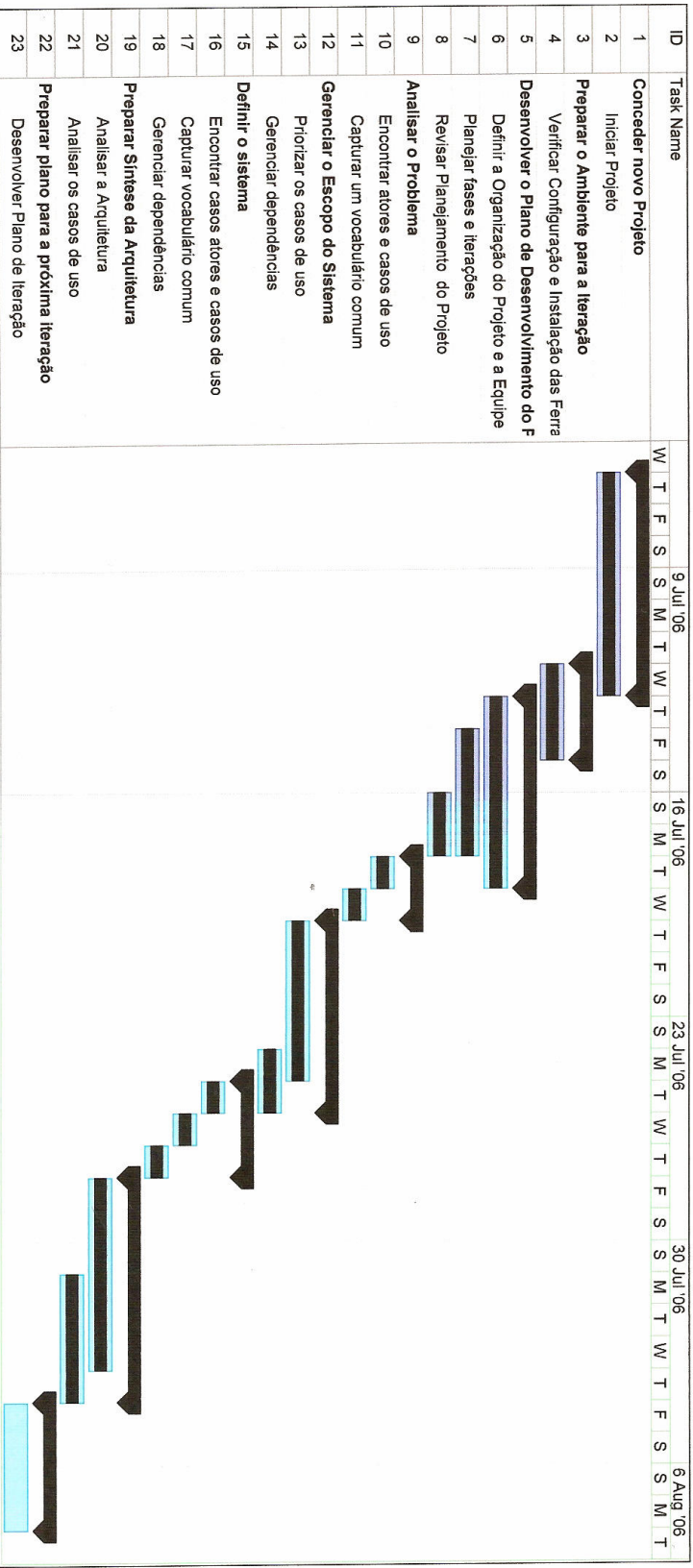
Estão descritas no Glossário (APÊNDICE J).

#### 1.3 Visão Geral

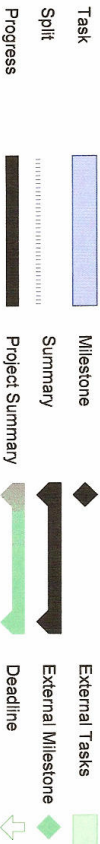
Este documento descreve o plano inicial do projeto através de um cronograma feito com o auxílio do *Microsoft Project*. No cronograma estão descritas as datas de início e fim das atividades inerentes a essa fase do projeto.

Outro ponto descrito no cronograma quando inicia-se a próxima fase que abrange a Elaboração do que foi analisado nessa primeira parte.

### 2. Cronograma



Project: concepcao  
Date: Mon 13/11/06





---

**Luciane Werlang & Jefferson de Oliveira**

---

**Sistema Manutenção  
Iteration Plan  
Plano de Iteração (Elaboração)**

**Versão 1.1**

<b>Disciplina:</b>	<i>Project Management</i> - Gerência de Projeto
<b>Papel:</b>	<i>Project Manager</i> - Gerente de Projeto
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Elaboração).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
13/06/2006	1.0	Edição inicial	Luciane e Jefferson
05/11/2006	1.1	Revisão Final	Luciane Werlang

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Elaboração).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
1.3	Visão Geral	4
2.	Cronograma	4

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Elaboração).doc	

# Iteration Plan

## Plano de Iteração (Elaboração)

### 1. Introdução

O documento propõe demonstrar um plano para a fase de elaboração do projeto.

#### 1.1 Objetivo

O objetivo é mostrar um cronograma contendo todas as atividades associadas a essa fase, juntamente com a datas e seus respectivos responsáveis.

#### 1.2 Definições, Acrônimos e Abreviações.

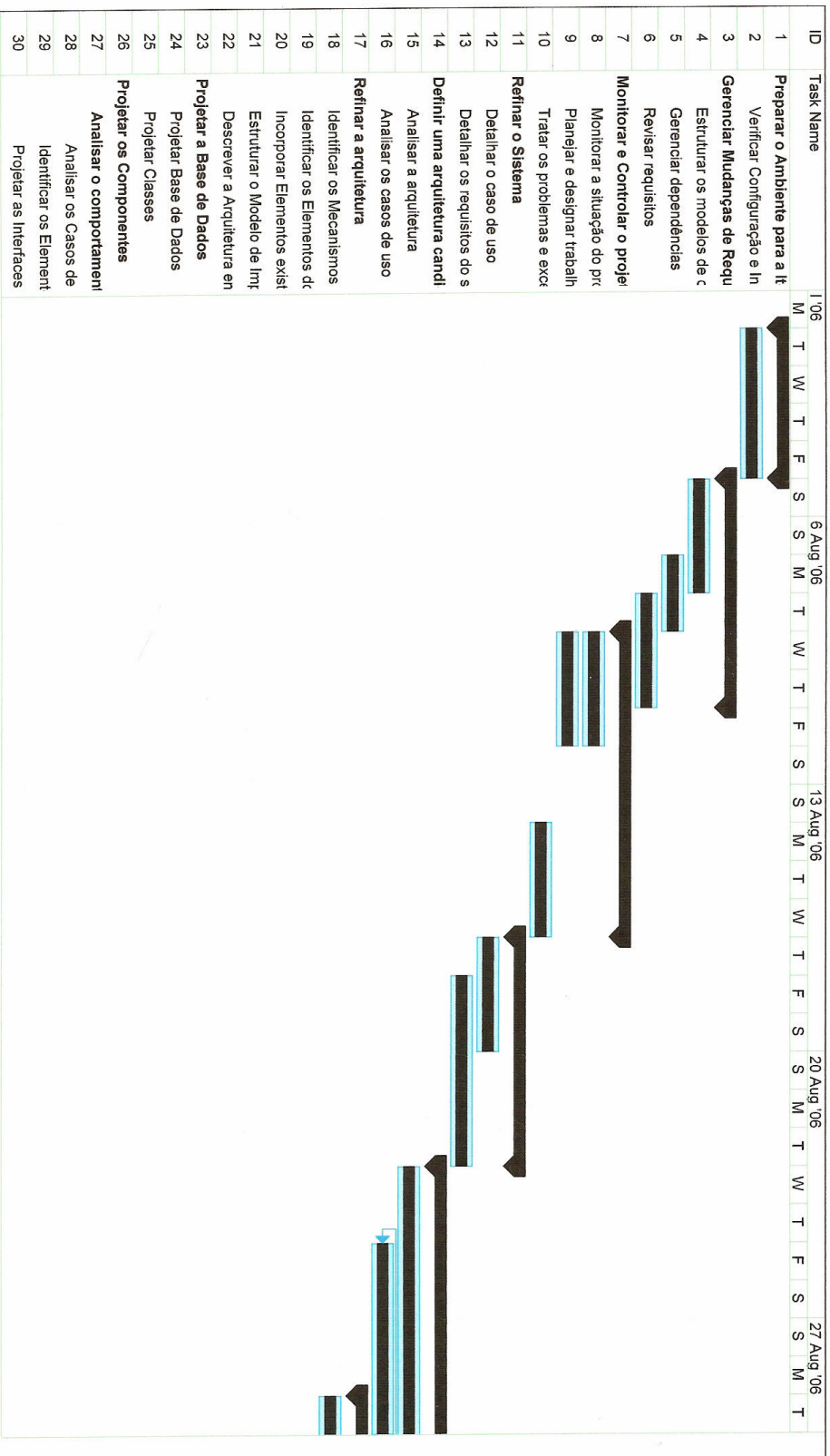
Estão descritas no Glossário (APÊNDICE J).

#### 1.3 Visão Geral

Este documento descreve o plano inicial do projeto através de um cronograma feito com o auxílio do *Microsoft Project*. No cronograma estão descritas as datas de início e fim das atividades inerentes a essa fase do projeto. Além disso, está descrito as atividades referentes ao componente que é desenvolvido para ser utilizado pelo sistema.

Outro ponto descrito no cronograma quando inicia-se a próxima fase que abrange a Construção do que foi analisado nas fases anteriores.

### 2. Cronograma



Project: Elaboração.mpp  
Date: Wed 15/11/06

Task

Split

Progress

Milestone

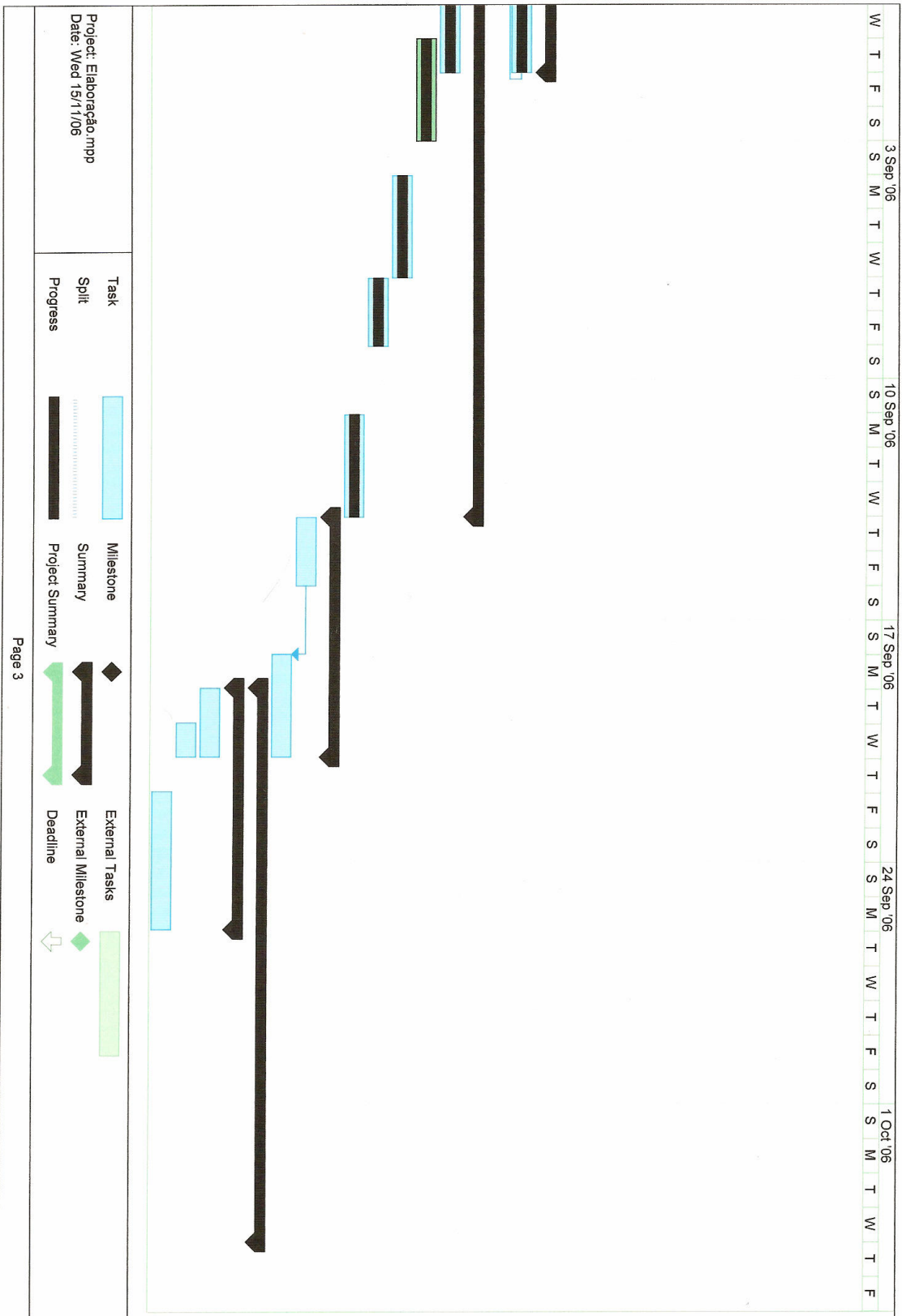
Summary

Project Summary

External Tasks

External Milestone

Deadline



ID	Task Name	106	6 Aug '06	13 Aug '06	20 Aug '06	27 Aug '06
31	Projetar Componentes	M	T	W	T	F
32	Projetar os casos de t					
33	Projetar as Classes					
34	Projetar Subsystemas					
35	Projetar módulos					
36	Integrar os componentes					
37	Integrar os subsystemas					
38	Testes e avaliações					
39	Executar os testes					
40	Próximo plano de iteração					
41	Desenvolver plano de itera					

Task

Split

Progress

Milestone

Summary

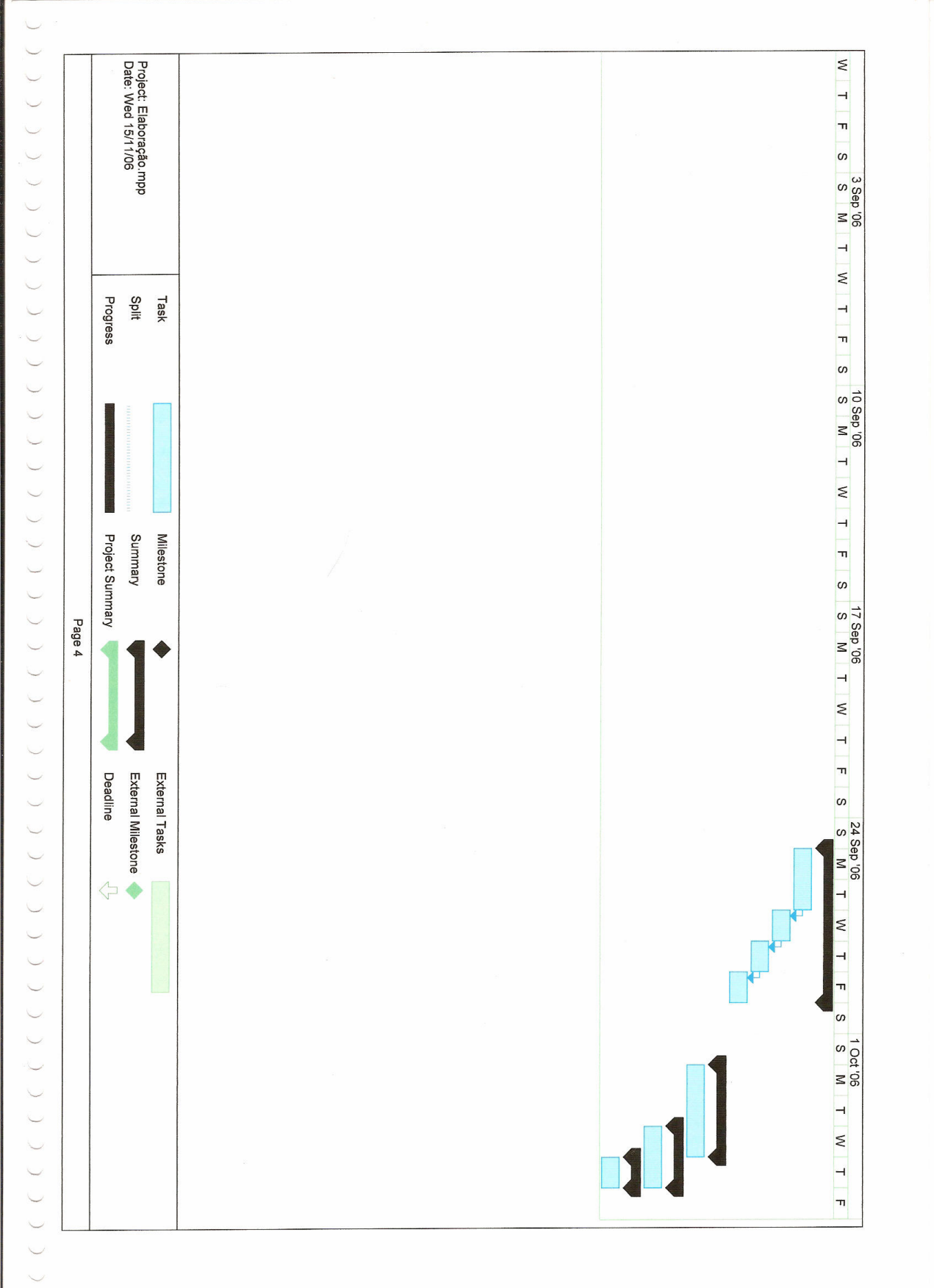
Project Summary

External Tasks

External Milestone

Deadline

Project: Elaboração.mpp  
Date: Wed 15/11/06





---

**Luciane Werlang & Jefferson de Oliveira**

---

**Sistema Manutenção  
Iteration Plan  
Plano de Iteração (Construção)**  
**Versão 1.1**

<b>Disciplina:</b>	<i>Project Management</i> - Gerência de Projeto
<b>Papel:</b>	<i>Project Manager</i> - Gerente de Projeto
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Construção).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
13/06/2006	1.0	Edição inicial	Luciane e Jefferson
05/11/2006	1.1	Revisão Final	Luciane Werlang

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Construção).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
1.3	Visão Geral	4
2.	Cronograma	4

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Construção).doc	

# Iteration Plan

## Plano de Iteração (Construção)

### 1. Introdução

O documento propõe demonstrar um plano para a fase de construção do projeto.

#### 1.1 Objetivo

O objetivo é mostrar um cronograma contendo todas as atividades associadas a essa fase, juntamente com a datas e seus respectivos responsáveis.

#### 1.2 Definições, Acrônimos e Abreviações.

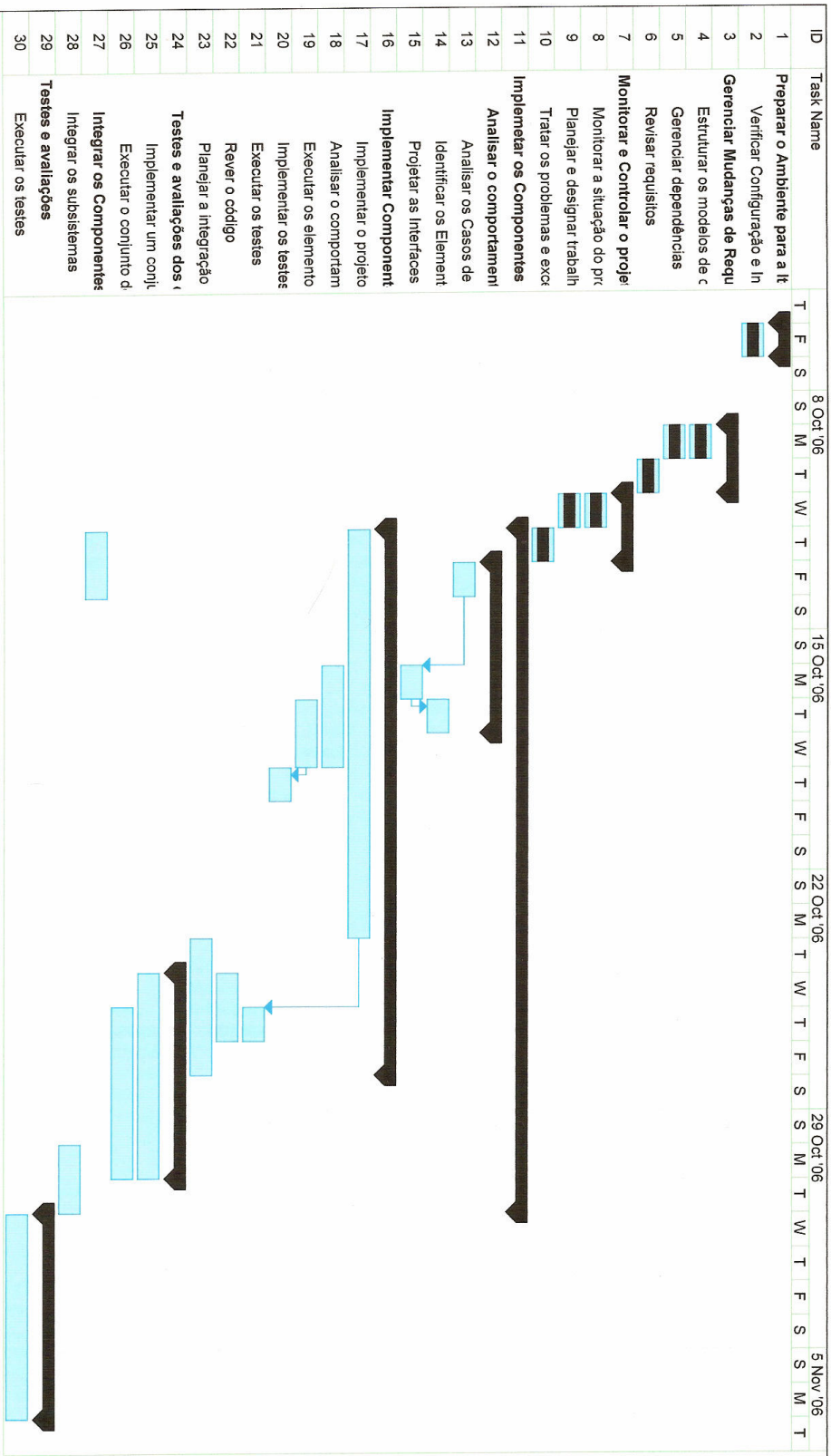
Estão descritas no Glossário (APÊNDICE J).

#### 1.3 Visão Geral

Este documento descreve o plano inicial do projeto através de um cronograma feito com o auxílio do *Microsoft Project*. No cronograma estão descritas as datas de início e fim das atividades inerentes a essa fase do projeto. Sendo uma fase que envolve o desenvolvimento do sistema e do componente, a parte de testes é bastante utilizada.

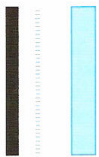
Outro ponto descrito no cronograma envolve o início da próxima fase que abrange a Transição, no qual é feita a implantação do sistema e resolvido alguns pontos que foram deixados de ser implementados e/ou não atenderam o que realmente tinha sido requisitado.

### 2. Cronograma



Project: Construção mpp  
Date: Wed 15/11/06

Task  
Split  
Progress



Milestone  
Summary



External Tasks  
External Milestone  
Deadline



ID	Task Name	8 Oct '06		15 Oct '06		22 Oct '06		29 Oct '06		5 Nov '06				
31	Próximo plano de Iteração	T	F	S	S	M	T	W	T	F	S	S	M	T
32	Desenvolver plano de itera													

Project: Construção mpp Date: Wed 15/1/06	Task Split Progress	Milestone Summary Project Summary	External Tasks External Milestone Deadline
--	---------------------------	---	--

---

**Luciane Werlang & Jefferson de Oliveira**

---

**Sistema Manutenção**  
**Iteration Plan**  
**Plano de Iteração (Transição)**  
**Versão 1.1**

<b>Disciplina:</b>	<i>Project Management</i> - Gerência de Projeto
<b>Papel:</b>	<i>Project Manager</i> - Gerente de Projeto
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Transição).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
13/06/2006	1.0	Edição inicial	Luciane e Jefferson
05/11/2006	1.1	Revisão Final	Luciane Werlang



Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Transição).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
1.3	Visão Geral	4
2.	Cronograma	4

Sistema Manutenção	Versão: 1.1
Iteration Plan	Data de Criação: 28-mai-2006
Arquivo: F - Plano de Iteração (Transição).doc	

# Iteration Plan

## Plano de Iteração (Transição)

### 1. Introdução

O documento propõe demonstrar um plano para a fase de transição do projeto.

#### 1.1 Objetivo

O objetivo é mostrar um cronograma contendo todas as atividades associadas a essa fase, juntamente com as datas e seus respectivos responsáveis.

#### 1.2 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

#### 1.3 Visão Geral

Este documento descreve o plano inicial do projeto através de um cronograma feito com o auxílio do *Microsoft Project*. No cronograma estão descritas as datas de início e fim das atividades inerentes a essa fase do projeto.

### 2. Cronograma

ID	Task Name	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M
1	Preparar o Ambiente para a Iteração																											
2	Verificar Configuração e Instalação das Ferrament																											
3	Implementar os Componentes (Defeitos)																											
4	Rever código																											
5	Analisar o comportamento da aplicação em execu																											
6	Executar elementos de testabilidade																											
7	Implementar testes																											
8	Executar testes																											
9	Planejar a Integração do Subistema																											
10	Teste e avaliações dos Componentes																											
11	Executar os testes																											
12	Integrar ao Sistema																											
13	Integrar o sistema																											
14	Testes e avaliações																											
15	Executar os testes																											
16	Desenvolver material de suporte																											
17	Desenvolver material de suporte																											
18	Finalizar o projeto																											
19	Preparar para finalizar o projeto																											

Task

Split

Progress

Milestone

Summary

Project Summary

External Tasks

External Milestone

Deadline

Project: Transição.mpp

Date: Wed 15/1/06

Page 1

## **APÊNDICE G – Artefato do RUP de Especificação da Realização do Caso de Uso**

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Use-Case-Realization Specification: Manutenção de**  
**País**  
**Especificação da Realização do Caso de Uso**

**Versão 1.1**

<b>Disciplina:</b>	<i>Analysis &amp; Design</i> - Análise e Design
<b>Papel:</b>	<i>Designer</i> - Designer
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
03/06/2006	1.0	Edição Inicial	Jefferson Oliveira
05/11/2006	1.1	Revisão Final	Luciane Werlang

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
2.	Fluxo de Eventos – Design	4
3.	Realização do Caso de Uso	4
4.	Diagrama de Classes para Análise	5
5.	Diagramas de Seqüência	6
5.1	Atualizar	6
5.2	Consultar	7
5.3	Excluir	8
5.4	Incluir	9

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

# Use-Case-Realization Specification: Manutenção de País

## Especificação da Realização do Caso de Uso

### 1. Introdução

Este caso de uso tem por finalidade realizar a manutenção de dados do país.

#### 1.1 Objetivo

Realizar e manter o cadastro de Países.

#### 1.2 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

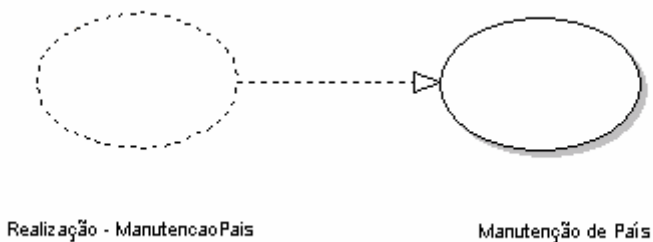
### 2. Fluxo de Eventos – Design

Iniciado por: Ator Usuário

O usuário poderá cadastrar novos Países quando clicar no botão novo da tela de consulta. Para realizar a alteração ou exclusão o usuário deverá realizar primeiramente a consulta de países e então selecionar um para que o mesmo seja editado. Nesse momento o usuário poderá escolher se quer excluir ou alterar.

### 3. Realização do Caso de Uso

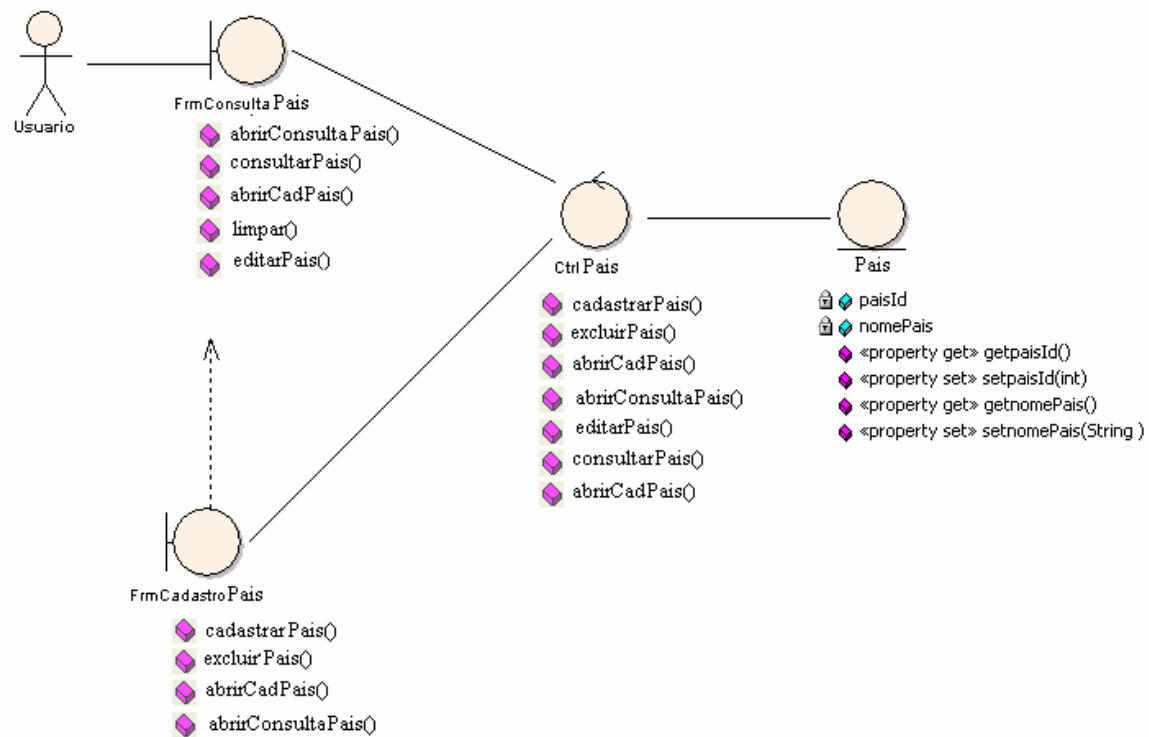
Representa a relação entre o modelo conceitual e o modelo de implementação.





Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

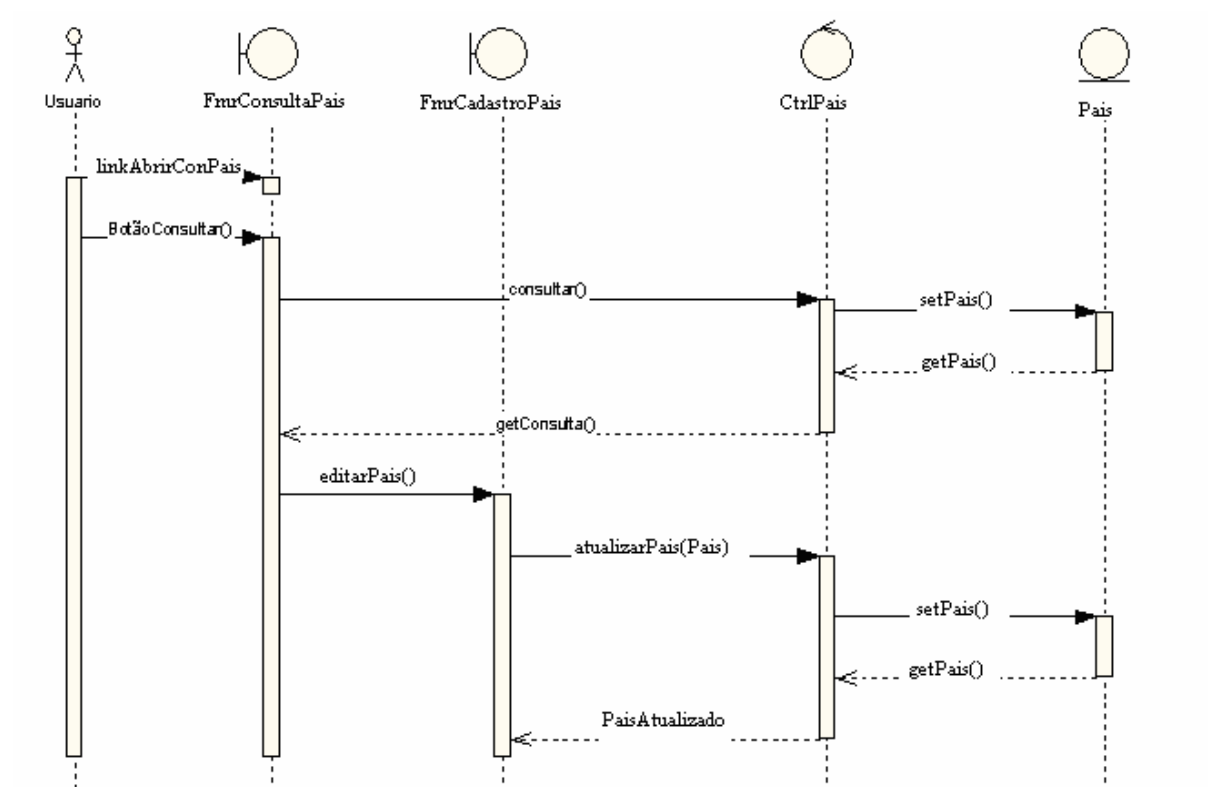
#### 4. Diagrama de Classes para Análise



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

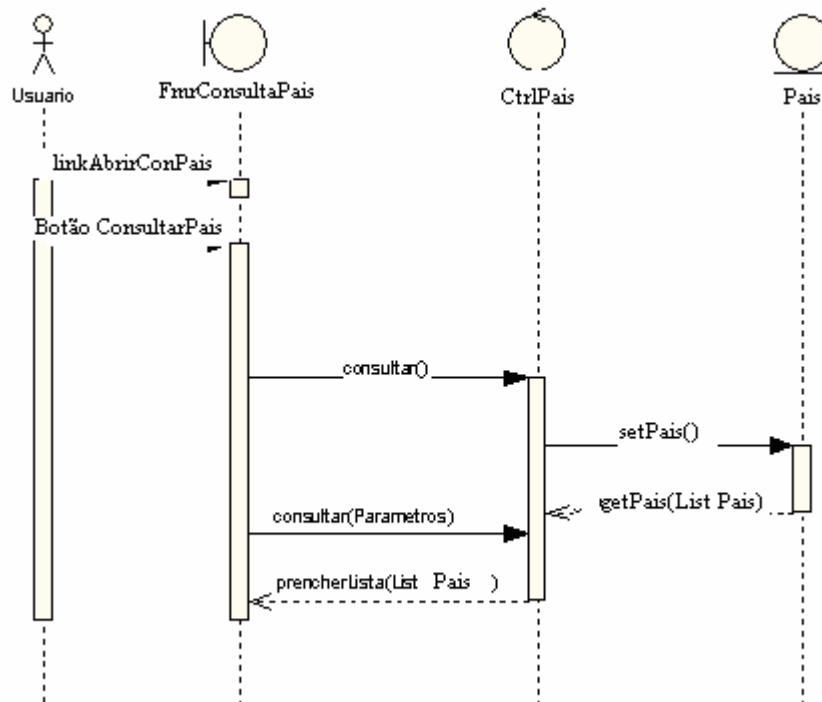
## 5. Diagramas de Seqüência

### 5.1 Atualizar



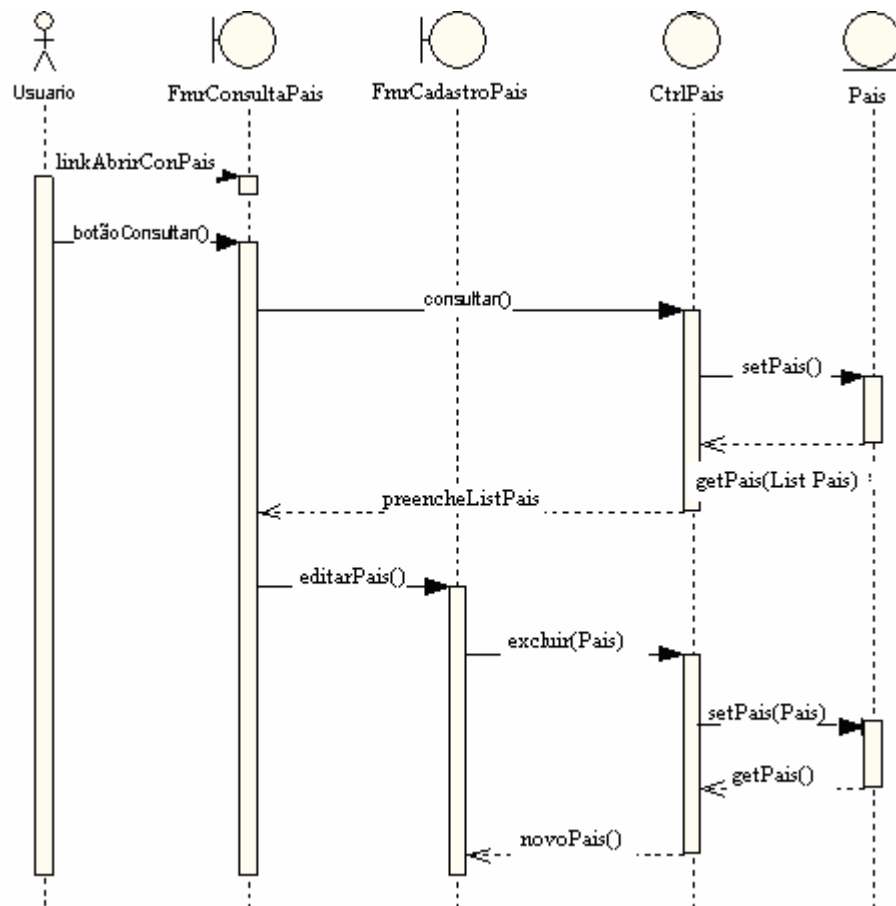
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

## 5.2 Consultar



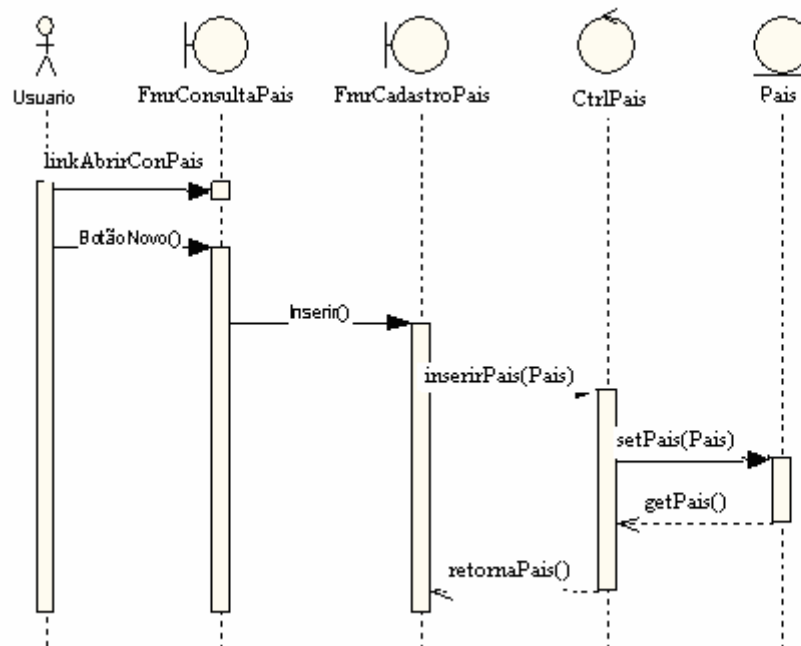
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

### 5.3 Excluir



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de País	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de País).doc	

#### 5.4 Incluir



---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Use-Case-Realization Specification: Manutenção de**  
**Estado**  
**Especificação da Realização do Caso de Uso**

**Versão 1.1**

<b>Disciplina:</b>	<i>Analysis &amp; Design</i> - Análise e Design
<b>Papel:</b>	<i>Designer</i> - Designer
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
03/06/2006	1.0	Edição Inicial	Jefferson Oliveira
05/11/2006	1.1	Revisão Final	Luciane Werlang

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
2.	Fluxo de Eventos – Design	4
3.	Realização do Caso de Uso	4
4.	Diagrama de Classes para Análise	5
5.	Diagramas de Seqüência	6
5.1	Atualizar	6
5.2	Consultar	7
5.3	Excluir	8
5.4	Incluir	9



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

# Use-Case-Realization Specification: Manutenção de Estado

## Especificação da Realização do Caso de Uso

### 1. Introdução

Este caso de uso tem por finalidade realizar a manutenção de dados do estado.

#### 1.1 Objetivo

Realizar e manter o cadastro de Estado.

#### 1.2 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

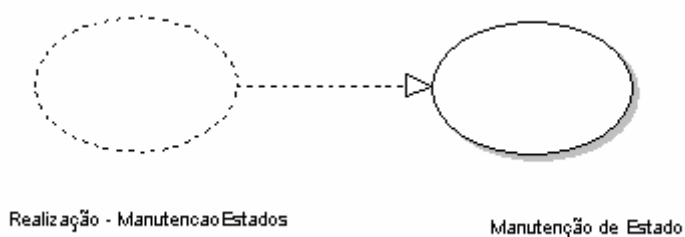
### 2. Fluxo de Eventos – Design

Iniciado por: Ator Usuário

O usuário poderá cadastrar novos Estados quando clicar no botão novo da tela de consulta. Para realizar a alteração ou exclusão o usuário deverá realizar primeiramente a consulta de estados e então selecionar um para que o mesmo seja editado. Nesse momento o usuário poderá escolher se quer excluir ou alterar.

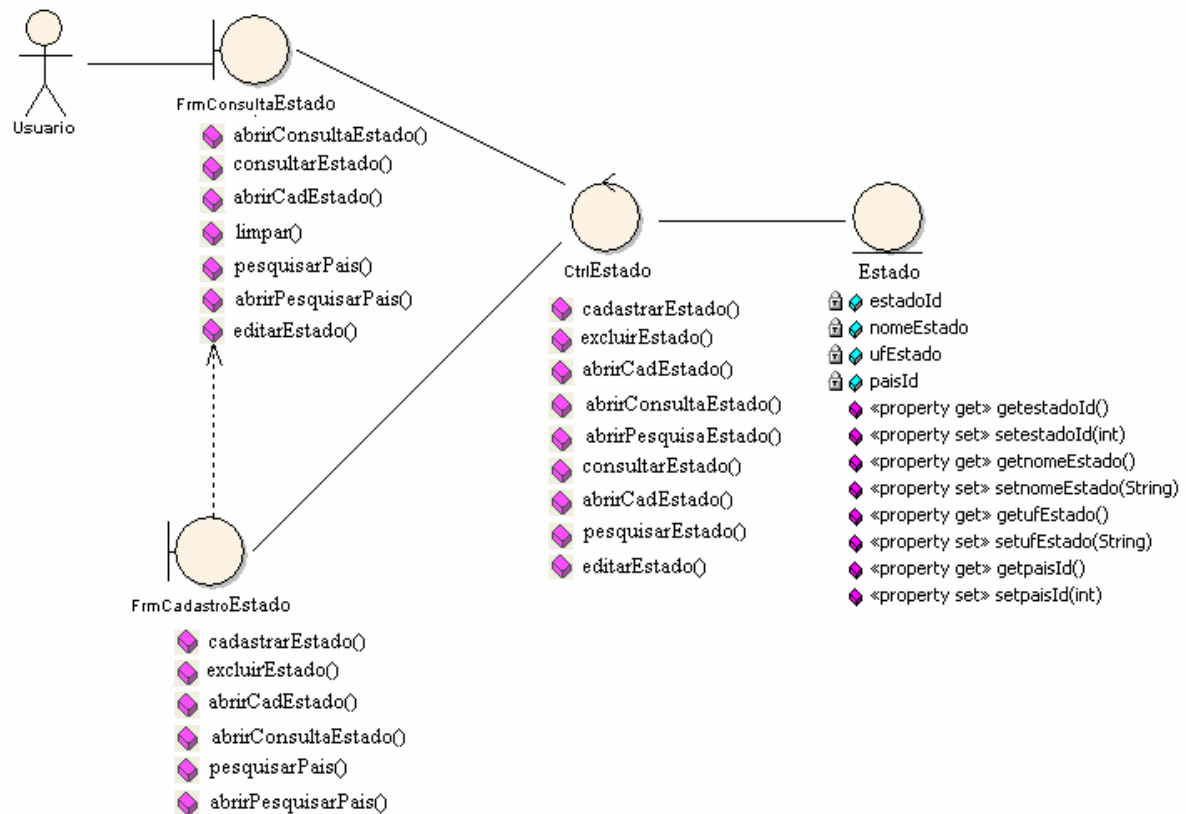
### 3. Realização do Caso de Uso

Representa a relação entre o modelo conceitual e o modelo de implementação.



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

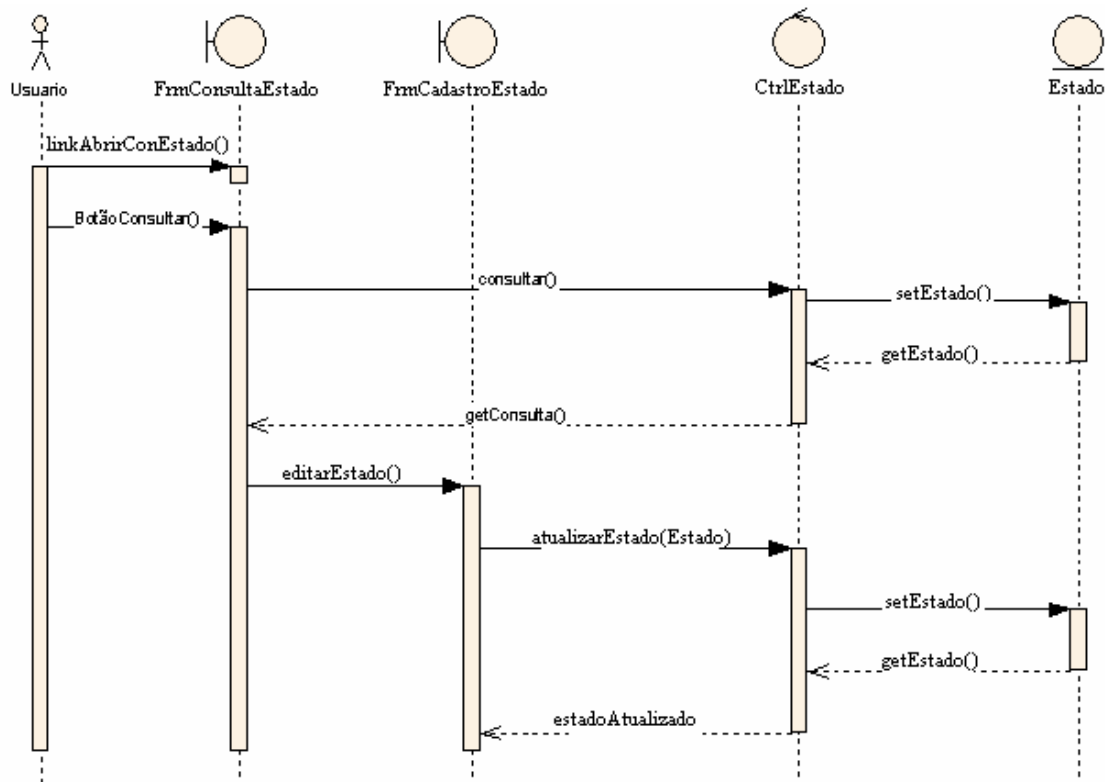
#### 4. Diagrama de Classes para Análise



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

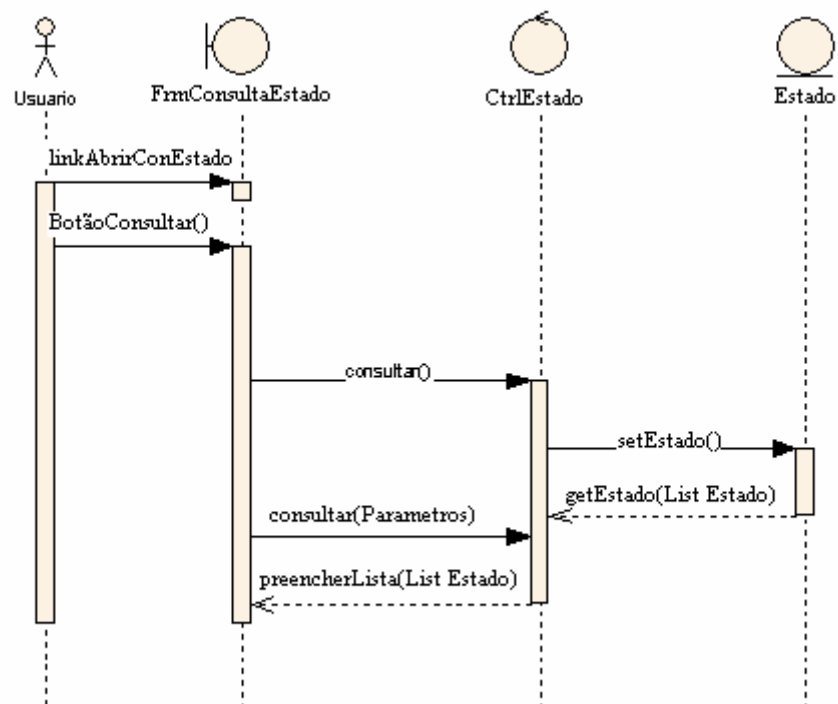
## 5. Diagramas de Sequência

### 5.1 Atualizar



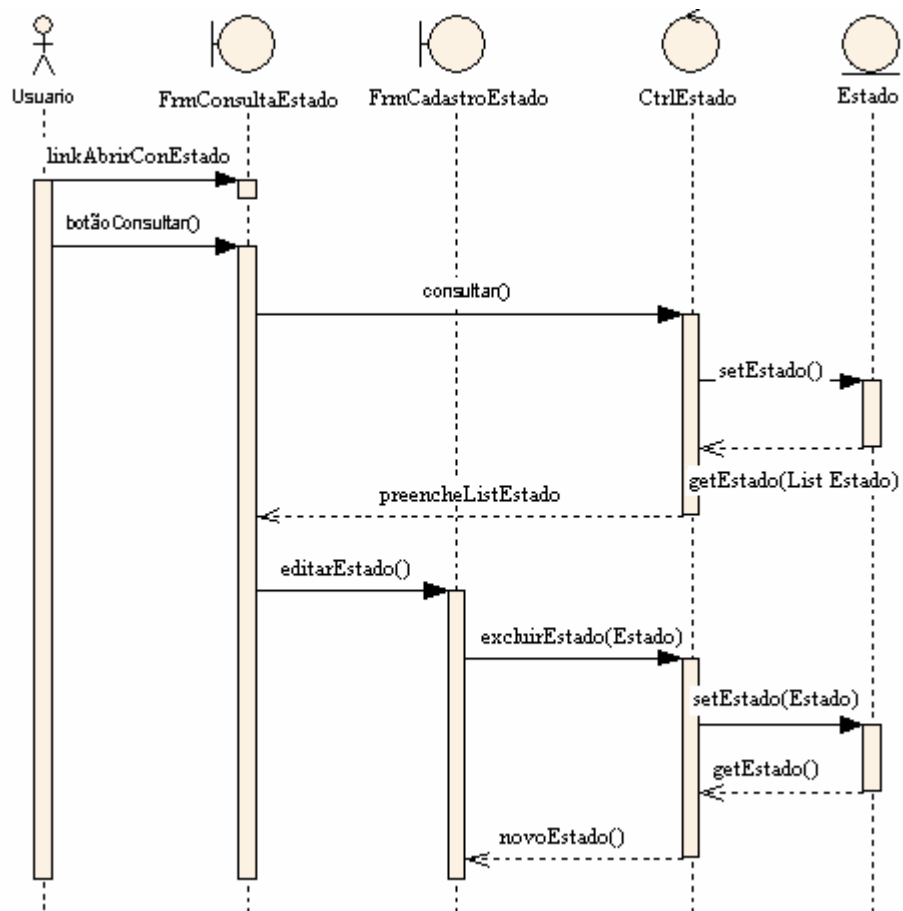
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

## 5.2 Consultar



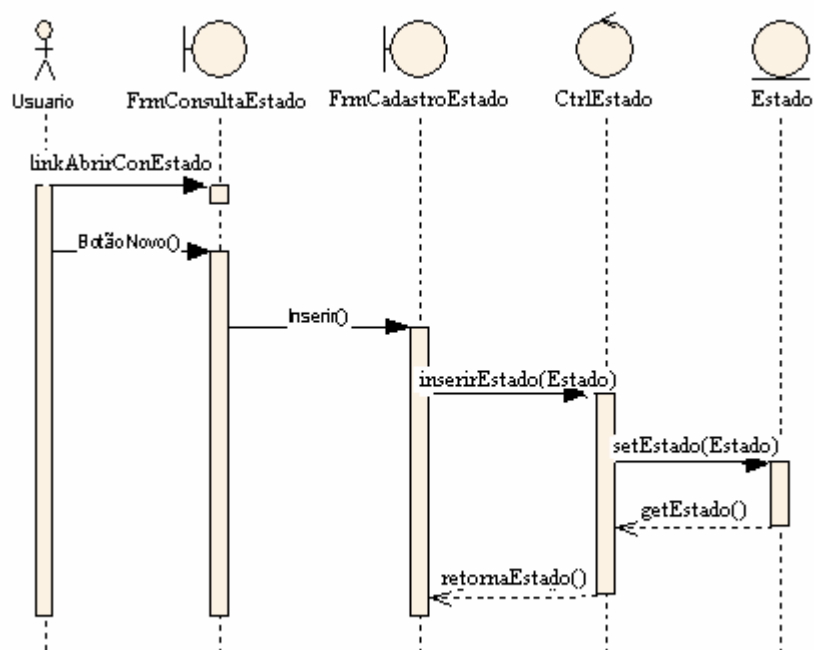
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

### 5.3 Excluir



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Estado	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Estado).doc	

#### 5.4 Incluir



---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Use-Case-Realization Specification: Manutenção de**  
**Cidade**  
**Especificação da Realização do Caso de Uso**

**Versão 1.1**

<b>Disciplina:</b>	<i>Analysis &amp; Design</i> - Análise e Design
<b>Papel:</b>	<i>Designer</i> - Designer
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
03/06/2006	1.0	Edição Inicial	Jefferson Oliveira
05/11/2006	1.1	Revisão Final	Luciane Werlang



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
2.	Fluxo de Eventos – Design	4
3.	Realização do Caso de Uso	4
4.	Diagrama de Classes para Análise	5
5.	Diagramas de Seqüência	6
5.1	Atualizar	6
5.2	Consultar	7
5.3	Excluir	8
5.4	Incluir	9

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

# Use-Case-Realization Specification: Manutenção de Cidade

## Especificação da Realização do Caso de Uso

### 1. Introdução

Este caso de uso tem por finalidade realizar a manutenção de dados da cidade.

#### 1.1 Objetivo

Realizar e manter o cadastro de Cidades.

#### 1.2 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

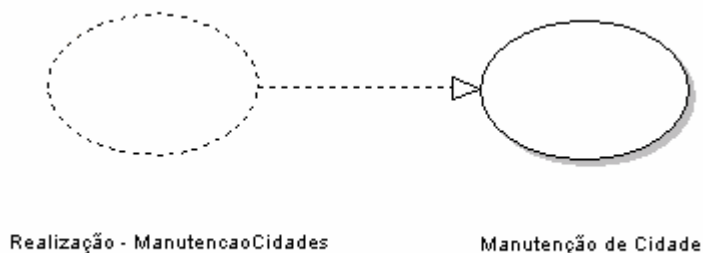
### 2. Fluxo de Eventos – Design

Iniciado por: Ator Usuário

O usuário poderá cadastrar novas Cidades quando clicar no botão novo da tela de consulta. Para realizar a alteração ou exclusão o usuário deverá realizar primeiramente a consulta de cidades e então selecionar um para que o mesmo seja editado. Nesse momento o usuário poderá escolher se quer excluir ou alterar.

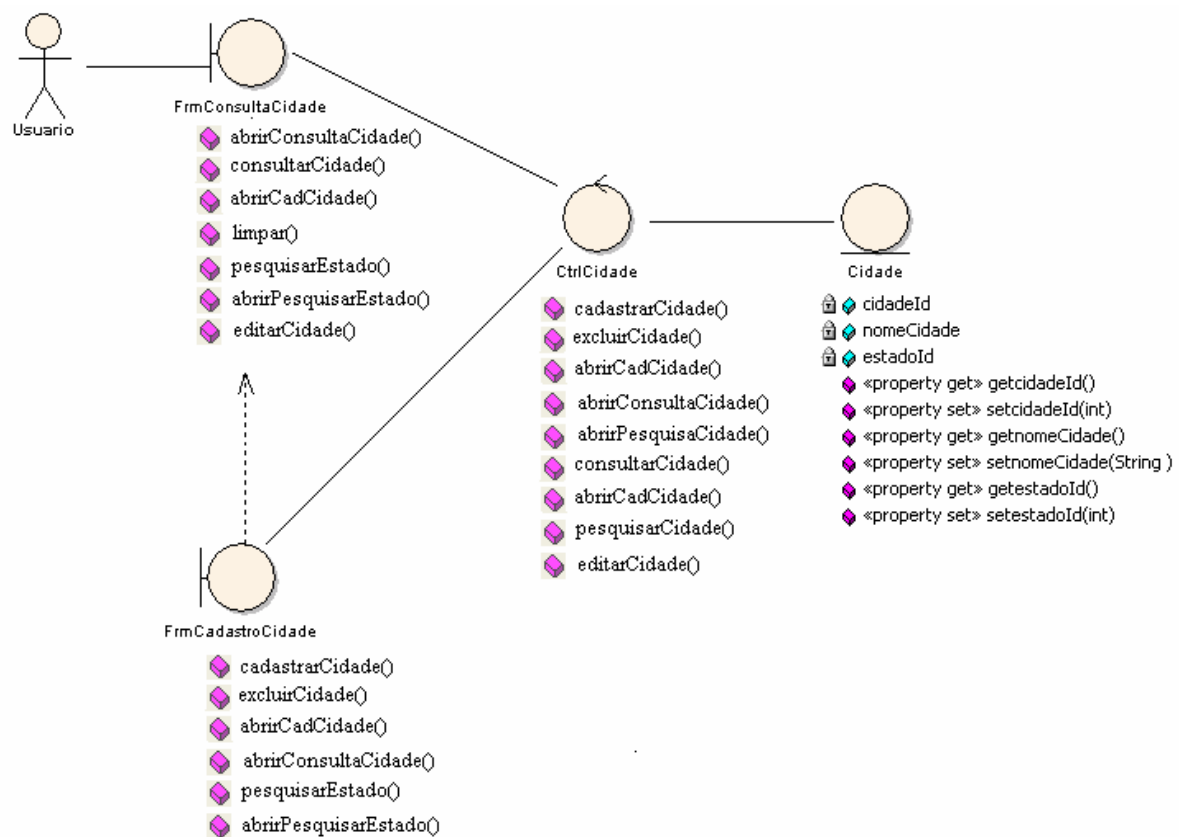
### 3. Realização do Caso de Uso

Representa a relação entre o modelo conceitual e o modelo de implementação.



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

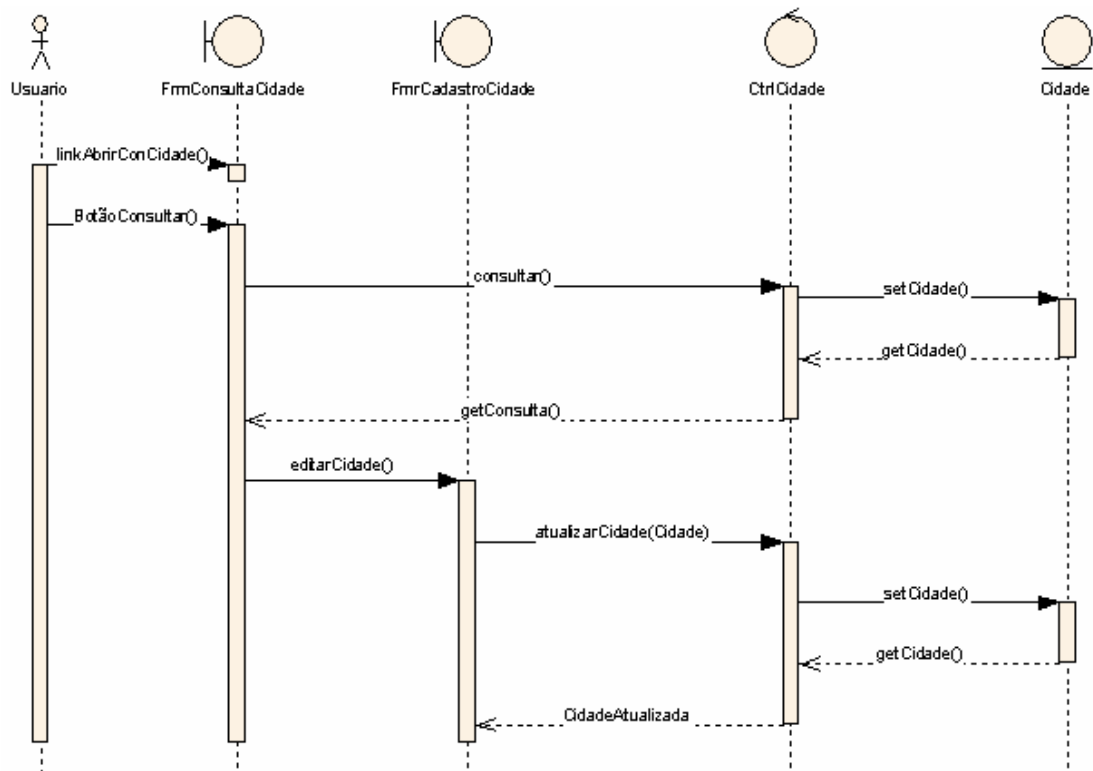
#### 4. Diagrama de Classes para Análise



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

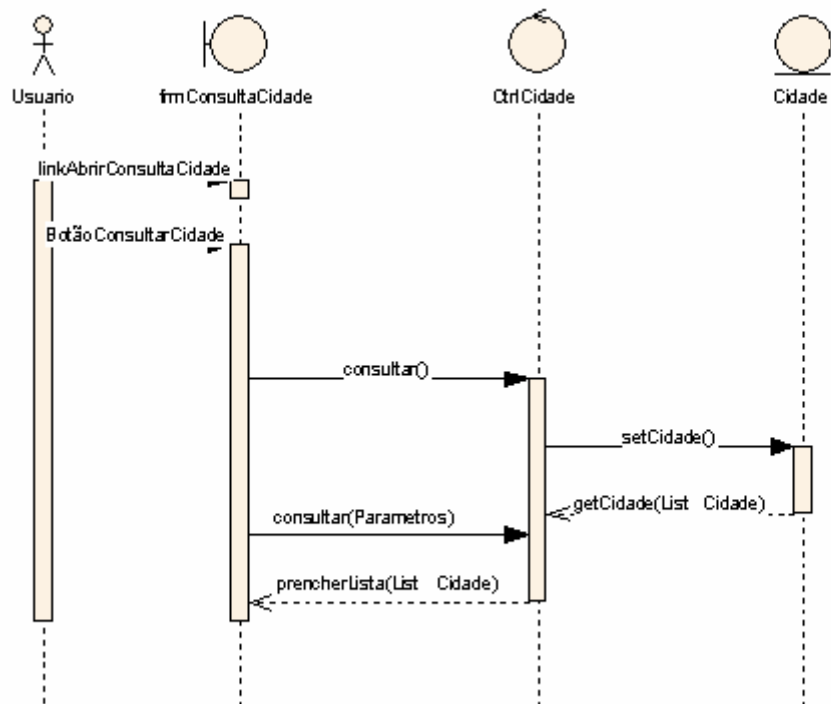
## 5. Diagramas de Seqüência

### 5.1 Atualizar



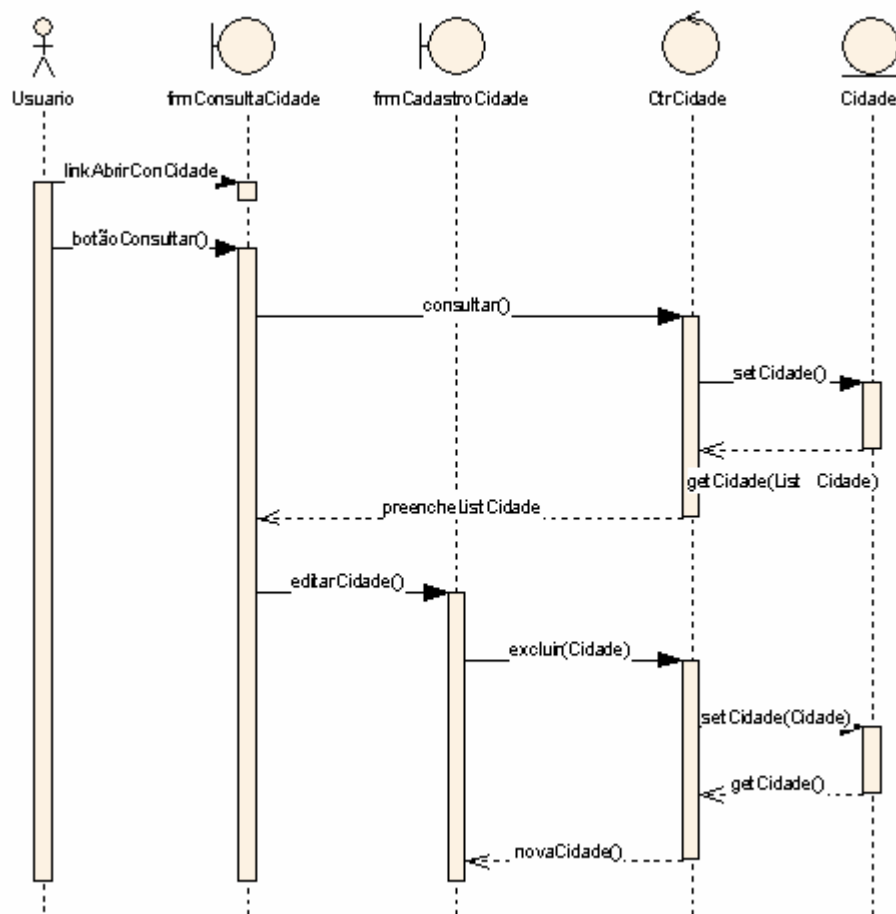
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

## 5.2 Consultar



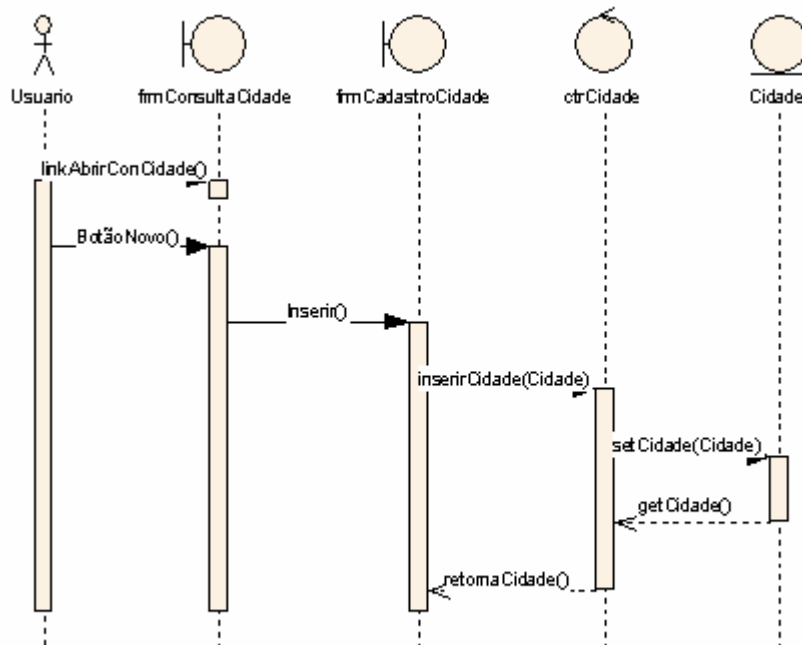
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

### 5.3 Excluir



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cidade	Data de Criação: 19-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cidade).doc	

## 5.4 Incluir



---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Use-Case-Realization Specification: Manutenção de**  
**Cliente**  
**Especificação da Realização do Caso de Uso**

**Versão 1.1**

<b>Disciplina:</b>	<i>Analysis &amp; Design</i> - Análise e Design
<b>Papel:</b>	<i>Designer</i> - Designer
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
03/06/2006	1.0	Edição Inicial	Jefferson Oliveira
05/11/2006	1.1	Revisão Final	Luciane Werlang

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Definições, Acrônimos e Abreviações.	4
2.	Fluxo de Eventos – Design	4
3.	Realização do Caso de Uso	4
4.	Diagrama de Classes para Análise	5
5.	Diagramas de Seqüência	6
5.1	Atualizar	6
5.2	Consultar	7
5.3	Excluir	8
5.4	Incluir	9

Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

# Use-Case-Realization Specification: Manutenção de Cliente

## Especificação da Realização do Caso de Uso

### 1. Introdução

Este caso de uso tem por finalidade realizar a manutenção de dados do Cliente.

#### 1.1 Objetivo

Realizar e manter o cadastro de Clientes.

#### 1.2 Definições, Acrônimos e Abreviações.

Estão descritas no Glossário (APÊNDICE J).

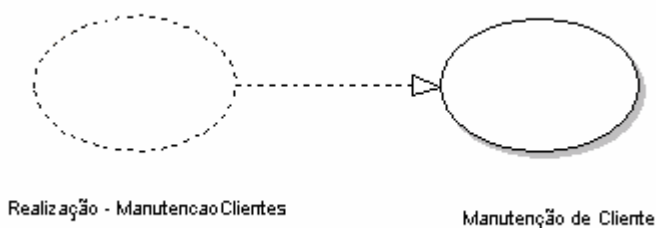
### 2. Fluxo de Eventos – Design

Iniciado por: Ator Usuário

O usuário poderá cadastrar novos Clientes quando clicar no botão novo da tela de consulta. Para realizar a alteração ou exclusão o usuário deverá realizar primeiramente a consulta de clientes e então selecionar um para que o mesmo seja editado. Nesse momento o usuário poderá escolher se quer excluir ou alterar.

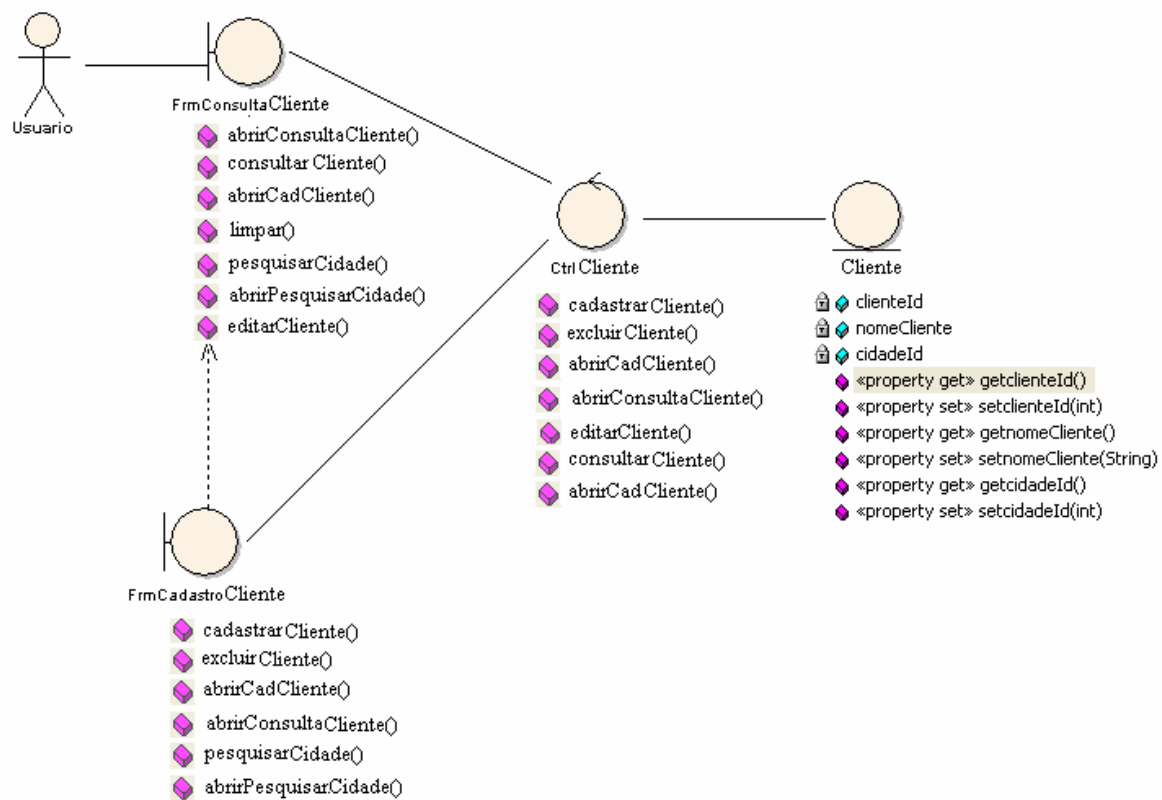
### 3. Realização do Caso de Uso

Representa a relação entre o modelo conceitual e o modelo de implementação.



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

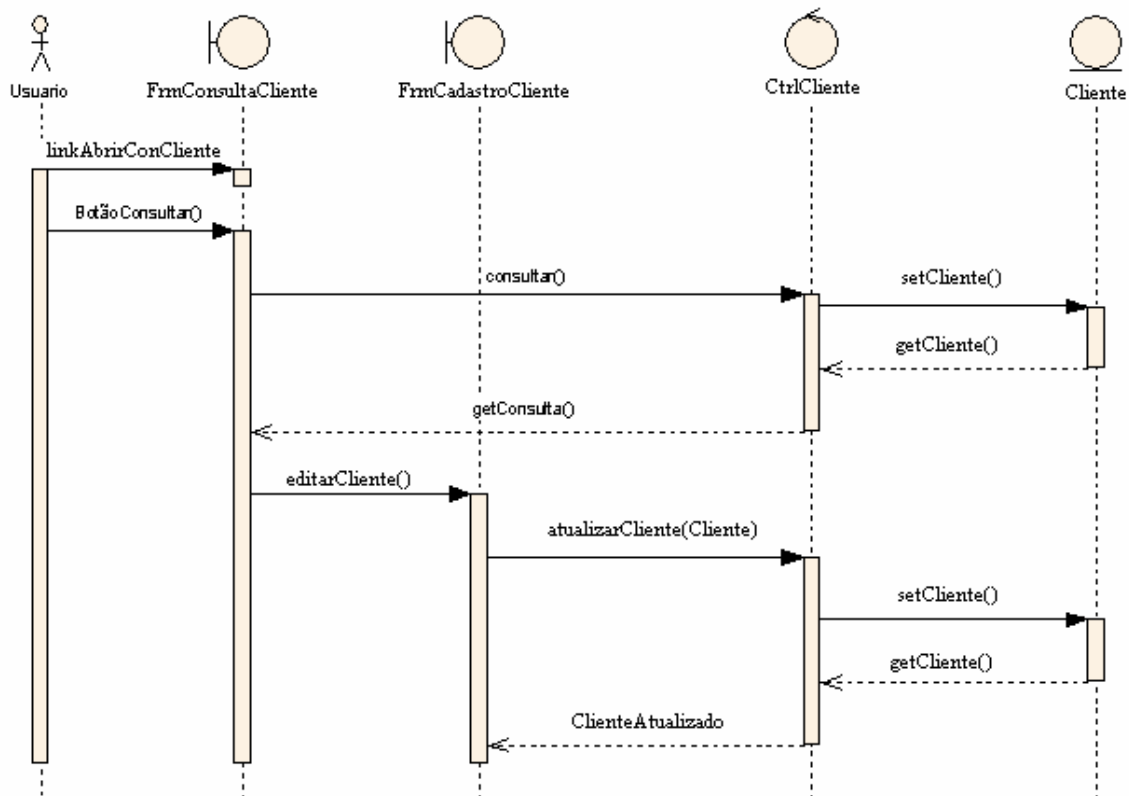
#### 4. Diagrama de Classes para Análise



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

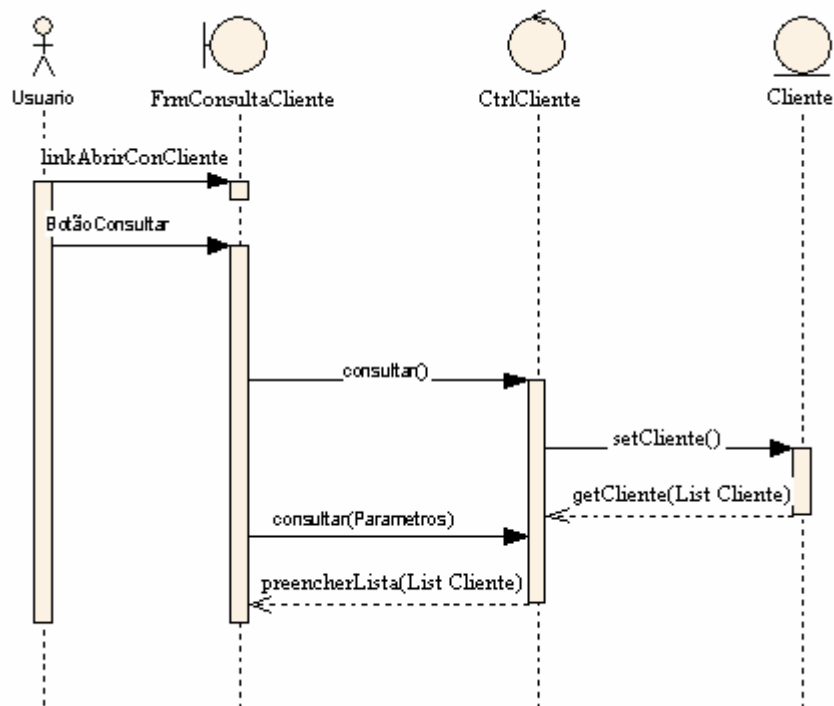
## 5. Diagramas de Seqüência

### 5.1 Atualizar



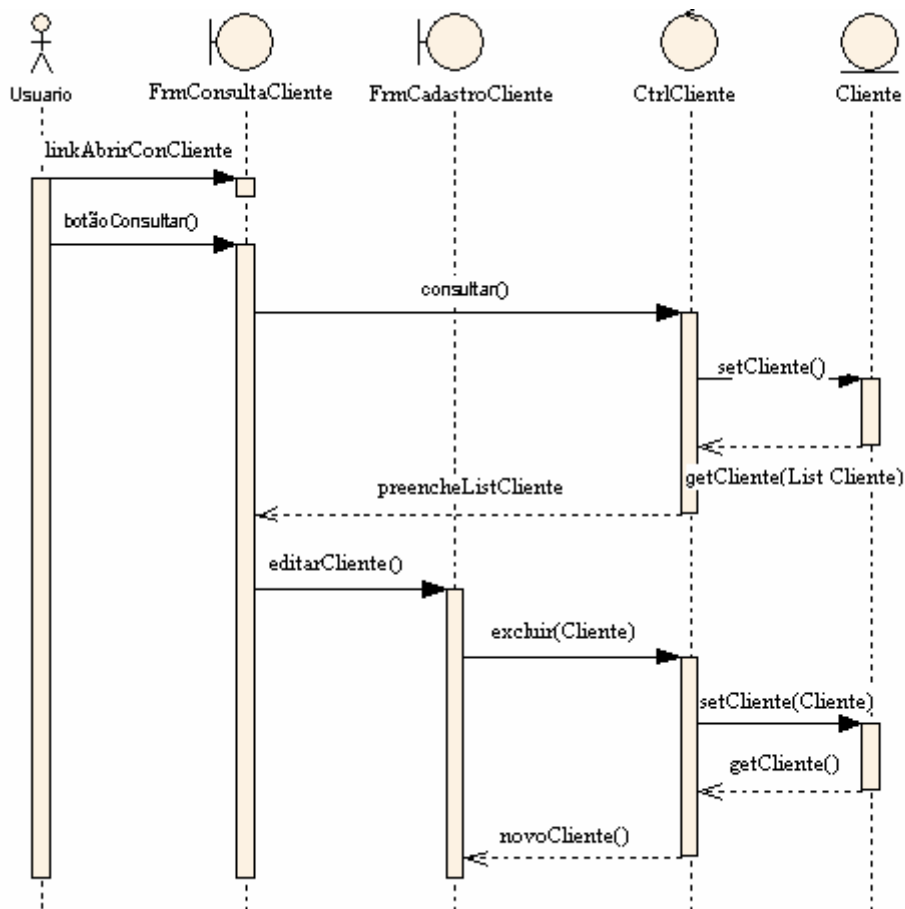
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

## 5.2 Consultar



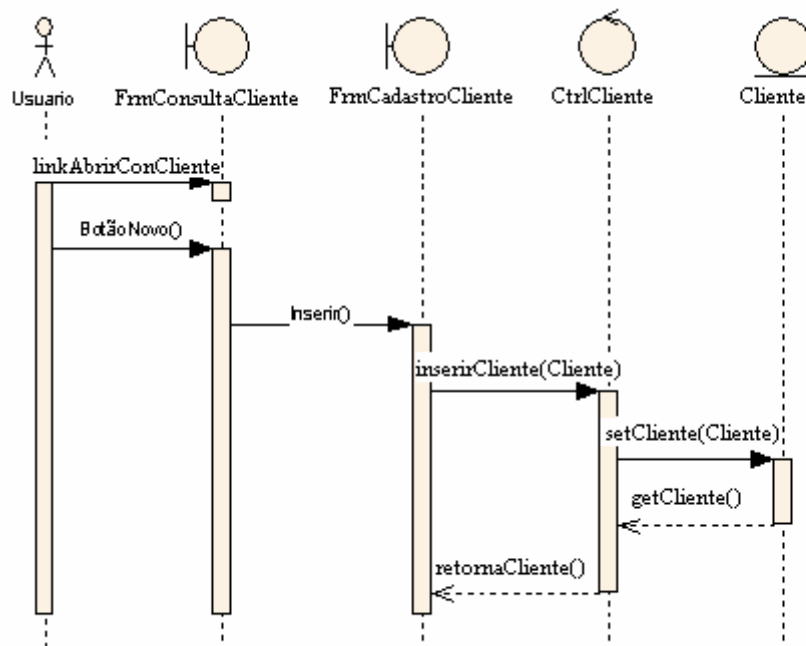
Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

### 5.3 Excluir



Sistema Manutenção	Versão: 1.1
Use-Case-Realization Specification: Manutenção de Cliente	Data de Criação: 25-jun-2006
Arquivo: G - Especificação da Realização do Caso de Uso (Manutenção de Cliente).doc	

#### 5.4 Incluir





## **APÊNDICE H – Artefato do RUP de Plano de Implementação**

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Integration Build Plan**  
**Plano de Construção de Integração**  
**Versão 1.1**

<b>Disciplina:</b>	<i>Implementation</i> - Implementação
<b>Papel:</b>	<i>Integrator</i> - Integrador
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Integration Build Plan	Data de Criação: 28-mai-2006
Arquivo: H - Plano de Implementação.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
10/09/2006	1.0	Edição Inicial	Luciane e Jefferson
05/11/2006	1.1	Revisão Final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.1
Integration Build Plan	Data de Criação: 28-mai-2006
Arquivo: H - Plano de Implementação.doc	

## Sumário

1.	Introdução	4
1.1	Definições, Acrônimos e Abreviações	4
1.2	Visão Geral	4
2.	Sub-Sistemas	4
3.	Construções	4

Sistema Manutenção	Versão: 1.1
Integration Build Plan	Data de Criação: 28-mai-2006
Arquivo: H - Plano de Implementação.doc	

# Integration Build Plan

## Plano de Construção de Integração

### 1. Introdução

Este documento inclui as definições, acrônimos, abreviações, a visão geral do **Build Plan (Plano de Construção de Integração)**.

#### 1.1 Definições, Acrônimos e Abreviações

Estão descritas no Glossário (APÊNDICE J).

#### 1.2 Visão Geral

Este Plano de Construção descreve os procedimentos adotados para iniciar a implementação do sistema.

### 2. Sub-Sistemas

O sistema que será implementado é composto por pacotes distintos. Esses pacotes são implementados na seguinte ordem: Pacote País, Pacote Estado, Pacote Cidade, Pacote Cliente.

### 3. Construções

Esta seção especifica quais construções criar e quais sub-sistemas farão parte de cada construção. Para cada construção é necessário especificar como a construção é feita, o critério para sua avaliação e como ele será testado, em particular:

- Manutenção de País

Construção

Foram gerados os *scripts* necessários para criar no banco de dados a tabela para desenvolver as operações de inclusão, alteração, exclusão e consulta.

Avaliação e Teste

A cada parte implementada foram feitos testes para ver se as operações estavam sendo atendidas.

- Manutenção do Estado

Construção

Foram gerados os *scripts* necessários para criar no banco de dados as tabelas para desenvolver as operações de inclusão, alteração, exclusão e consulta. Neste caso, o cadastro de estado necessitava de um País.

Avaliação e Teste

A cada parte implementada foram feitos testes para ver se as operações estavam sendo atendidas.

- Manutenção do Cidade

Construção

Foram gerados os *scripts* necessários para criar no banco de dados as tabelas para desenvolver as operações de inclusão, alteração, exclusão e consulta. Neste caso, o cadastro de estado necessitava de

Sistema Manutenção	Versão: 1.1
Integration Build Plan	Data de Criação: 28-mai-2006
Arquivo: H - Plano de Implementação.doc	

um Estado e um País.

#### Avaliação e Teste

A cada parte implementada foram feitos testes para ver se as operações estavam sendo atendidas.

- Manutenção do Cliente

#### Construção

Foram gerados os *scripts* necessários para criar no banco de dados as tabelas para desenvolver as operações de inclusão, alteração, exclusão e consulta. Neste caso, o cadastro de estado necessitava de um Estado e um País e uma Cidade.

#### Avaliação e Teste

A cada parte implementada foram feitos testes para ver se as operações estavam sendo atendidas.

- Pesquisar

#### Construção

Foram gerados os *scripts* necessários para criar no banco de dados as tabelas que serão utilizadas para realizar essa funcionalidade.

#### Avaliação e Teste

A cada parte implementada foram feitos testes para ver se as operações estavam sendo atendidas.

**APÊNDICE I – Artefato do RUP de Plano de Teste**

---

**Luciane Werlang & Jefferson Oliveira**

---

**Sistema Manutenção**  
**Test Plan**  
**Plano de Teste**  
**Versão 1.1**

<b>Disciplina:</b>	<i>Test</i> - Teste
<b>Papel:</b>	<i>Teste Manager</i> - Gerente de Teste
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira



Sistema Manutenção	Versão: 1.1
Test Plan	Data de Criação: 28-mai-2006
Arquivo: I - Plano de Teste.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
15/10/2006	1.0	Edição Inicial	Luciane e Jefferson
02/11/2006	1.1	Revisão Final	Luciane e Jefferson

Sistema Manutenção	Versão: 1.1
Test Plan	Data de Criação: 28-mai-2006
Arquivo: I - Plano de Teste.doc	

## Sumário

1.	Introdução	4
1.1	Objetivo	4
1.2	Identificação do Projeto	4
2.	Requisitos para Teste	4
3.	Estratégia de Teste	4
3.1	Tipos de Teste	4
3.1.1	Teste da Integridade de Dados e Banco de Dados	4
3.1.2	Teste de Função	5
3.1.3	Teste da Interface de Usuário	5
4.	Recursos	6
4.1	Cargos	6

Sistema Manutenção	Versão: 1.1
Test Plan	Data de Criação: 28-mai-2006
Arquivo: I - Plano de Teste.doc	

# Test Plan

## Plano de Teste

### 1. Introdução

O *Test Plan* (Plano de Teste) fornece uma visão geral sobre os testes realizados no sistema

#### 1.1 Objetivo

Descrever os testes realizados no sistema.

#### 1.2 Identificação do Projeto

A tabela a seguir identifica a documentação e disponibilidade usada para o desenvolvimento do *test plan* (*plano de teste*):

Documento (e versão / data)	Criado ou Disponível	Foi Aceito ou Analisado
Especificação de Requisitos	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Especificação Funcional	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Relatórios de Caso de Uso	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Plano de Projeto	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Especificação de Design	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Protótipo	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Manuais do Usuário	<input type="checkbox"/> Sim <input checked="" type="checkbox"/> Não	<input type="checkbox"/> Sim <input checked="" type="checkbox"/> Não
Modelo ou Fluxo de Dados	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não
Funções e Regras de Negócios	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não	<input checked="" type="checkbox"/> Sim <input type="checkbox"/> Não

### 2. Requisitos para Teste

A lista abaixo identifica aqueles itens – casos de uso, requisitos funcionais, e não-funcionais – que tem sido identificados como objetivo do teste. Esta lista representa o que será testado.

Manutenção de País;

Manutenção de Estado;

Manutenção de Cidade;

Manutenção de Cliente;

Pesquisar;

### 3. Estratégia de Teste

A Estratégia de Teste apresenta uma aproximação recomendada para o teste do alvo de teste. A seção anterior, Requisitos para Teste, descreveu o que será testado – isto descreve como o alvo de teste será testado.

#### 3.1 Tipos de Teste

##### 3.1.1 Teste da Integridade de Dados e Banco de Dados

Os bancos de dados e os processos de banco de dados deveriam ser testados como um subsistema dentro do Sistema Manutenção. Estes subsistemas deveriam ser testados sem a Interface de Usuário do Alvo de Teste como interface dos dados. Pesquisas adicionais dentro do Sistema de Gerenciamento de Banco de Dados (DBMS - Database Management System) precisam ser executadas para identificar as

Sistema Manutenção	Versão: 1.1
Test Plan	Data de Criação: 28-mai-2006
Arquivo: I - Plano de Teste.doc	

ferramentas e técnicas que devem existir para apoiar o teste identificado abaixo.

Objetivo de Teste:	Assegure que os métodos e processos de acesso ao banco de dados funcionem corretamente e sem corrupção (perda) de dados.
Técnica:	A cada função implementada será testada sua funcionalidade perante o banco de dados.
Critério de Finalização:	Todos os métodos e processos de acesso ao banco de dados funcionam como descritos e sem qualquer corrupção de dados.
Considerações Especiais:	A conexão com o banco de dados é feita via JDBC, sendo que a biblioteca é adquirida por meio de seus desenvolvedores.

### 3.1.2 Teste de Função

Teste de função pode ser traçado diretamente para os casos de uso ou funções de negócios e regras de negócios. As metas destes testes são verificar aceitação de dados adequados, processamento, retorno e a implementação apropriada das regras de negócio. Este tipo de teste é baseado na técnica de caixa preta; que é verificar a aplicação e seus processos internos interagindo com as aplicações via Gráficos da Interface do Usuário (GUI) e analisando as saídas ou resultados. A tabela a seguir identifica a descrição do teste recomendado para cada aplicação:

Objetivos de Teste:	Assegure uma correta funcionalidade do alvo de teste, incluindo navegação, entrada de dados, processamento, e retorno.
Técnica:	Para cada função implementada é realizado testes com diversos tipos de inserções e observar se o comportamento da aplicação é o esperado. Também serão realizados os mesmos testes para as exclusões, alterações e consultas.
Critério de Finalização:	<ul style="list-style-type: none"> <li>• Todos os testes planejados tem sido executados.</li> <li>• Todos os defeitos identificados tem sido enfocados.</li> </ul>
Considerações Especiais:	Os testes serão baseados em cima da aplicação.

### 3.1.3 Teste da Interface de Usuário

Teste da Interface de Usuário (UI) verifica uma interação do usuário com o software. O objetivo do teste de UI é ter certeza que a UI fornece ao usuário acesso e navegação apropriados através de funções do objetivo de teste. Além disso, teste de UI traz a certeza que os objetos dentro das funções de UI como esperado e conforme aos padrões da corporação ou indústria.

Objetivos de Teste:	<ul style="list-style-type: none"> <li>• A Navegação através do alvo de teste refletindo as funções de negócio e requisitos, incluindo janela para janela, campo para campo e o uso de métodos de acesso (tecla tab, movimentos do mouse)</li> <li>• Objetos de janela e características, podem ser exercitados – assim como menus, tamanhos, posições, estados e foco conforme padrão.</li> </ul>
Técnica:	<p>Verificar a ordem lógica dos botões nas telas;</p> <p>A ordem da tecla tab na navegação das telas;</p>

Sistema Manutenção	Versão: 1.1
Test Plan	Data de Criação: 28-mai-2006
Arquivo: I - Plano de Teste.doc	

Critério de Finalização:	Cada janela é verificada com sucesso para ficar consistente com versão de avaliação de execução ou dentro de padrão aceitável.
Considerações Especiais:	

## 4. Recursos

Esta seção apresenta os recursos recomendados para o projeto Sistema Manutenção, suas principais responsabilidades, e seus conhecimentos ou grupo de conhecimento profissional.

### 4.1 Cargos

Esta tabela mostra a concepção de grupo de trabalho para o projeto.

Trabalhador	Responsabilidades Específicas ou Comentários
Tester (Validador)	<p>Executa os testes.</p> <p>Responsabilidades:</p> <ul style="list-style-type: none"> <li>• Executa testes</li> <li>• Arquiva resultados</li> <li>• Recupera erros</li> <li>• Documenta requisitos de mudança</li> <li>• Administrar os dados de teste (banco de dados)</li> </ul>

**APÊNDICE J – Artefato do RUP de Glossário**

---

# Luciane Werlang & Jefferson de Oliveira

---

## Sistema Manutenção

### Glossary

### Glossário

### Versão 1.1

<b>Disciplina:</b>	<i>Requirements</i> - Requisitos
<b>Papel:</b>	<i>System Analyst</i> - Analista de Sistema
<b>Indivíduo:</b>	Luciane Pires Werlang & Jefferson Amorim de Oliveira

Sistema Manutenção	Versão: 1.1
Glossary	Data de Criação: 28-May-2006
Arquivo: J - Glossário.doc	

## Histórico de Revisões

<b>Data</b>	<b>Versão</b>	<b>Descrição</b>	<b>Autor</b>
13/06/2006	1.0	Edição inicial	Luciane & Jefferson
04/11/2006	1.1	Revisão Final	Jefferson Oliveira



Sistema Manutenção	Versão: 1.1
Glossary	Data de Criação: 28-May-2006
Arquivo: J - Glossário.doc	

# Sumário

1.	Introdução	4
2.	Definições	4
2.1	Apache Tomcat	4
2.2	Browser	4
2.3	Caso de Uso	4
2.4	Deploy	4
2.5	EJBs	4
2.6	GUI	4
2.7	HTML	4
2.8	Internet	4
2.9	Java	4
2.10	JavaScript	4
2.11	JSP	5
2.12	XML	5
2.13	DTD	5
2.14	Login	5
2.15	Protocolo HTTP	5
2.16	Protocolo TCP/IP	5
2.17	Servlet	5
2.18	Templates	5
2.19	UML	5

Sistema Manutenção	Versão: 1.1
Glossary	Data de Criação: 28-May-2006
Arquivo: J - Glossário.doc	

# Glossary

## Glossário

### 1. Introdução

Este documento é usado para definir nomenclaturas específicas ao domínio do problema, explicando os termos que podem não ser familiar ao leitor nas descrições dos Casos de Uso ou em outros documentos do projeto.

### 2. Definições

Os termos definidos aqui formam a substância essencial do documento.

#### 2.1 Apache Tomcat

Tomcat é um *Container Web*, parte da plataforma J2EE que abrange as tecnologias *Servlet* e *JSP*.

#### 2.2 Browser

Um navegador (também conhecido como *web browser* ou simplesmente *browser*) é um programa que habilita seus usuários a interagirem com documentos HTML hospedados em um servidor Web.

#### 2.3 Caso de Uso

É uma técnica para capturar os requisitos potenciais de um novo sistema ou mudança do sistema. Cada caso de uso fornece um ou mais cenários, que mostram como o sistema deve interagir com o usuário final ou um outro sistema, para conseguir um objetivo de negócio específico. Os casos de uso evitam tipicamente a linguagem técnica, preferindo a língua natural do usuário final.

#### 2.4 Deploy

A distribuição (*deploy*) do projeto.

#### 2.5 EJBs

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.

#### 2.6 GUI

Interface gráfica (GUI, do inglês *Graphical User Interface*) é um mecanismo de interação entre usuário e sistema de computador baseado em símbolos visuais, como ícones, menus e janelas.

#### 2.7 HTML

*HyperText Markup Language* (HTML) é uma linguagem de marcação utilizada para produzir páginas na Internet. Esses códigos podem ser interpretados pelos browsers para exibir as páginas da World Wide Web.

#### 2.8 Internet

A Internet é uma rede em escala mundial de milhões de computadores que permite o acesso a informações e todo tipo de transferência de dados.

#### 2.9 Java

É uma linguagem de programação orientada a objeto desenvolvida na década de 90 pelo programador James Gosling, na empresa *Sun Microsystems*. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual.

#### 2.10 JavaScript

*JavaScript* é uma linguagem de programação criada pela *Netscape*, para validação de formulários e interação com páginas da *web*.

Sistema Manutenção	Versão: 1.1
Glossary	Data de Criação: 28-May-2006
Arquivo: J - Glossário.doc	

## 2.11 JSP

*Java Server Pages* (JSP) é uma tecnologia para desenvolvimento de aplicações *WEB*. Permite que conteúdo dinâmico seja facilmente desenvolvido.

## 2.12 XML

*Extensible Markup Language* (XML) é um subtipo de *Standard Generalized Markup Language* - Linguagem Padronizada de Marcação Generica (SGML) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet.

## 2.13 DTD

Definição de Tipo de Documento (DTD), que serve para especificar quais elementos ou atributos são permitidos no documento XML e em que local do documento eles podem aparecer. Podemos definir, então, que o DTD é uma forma de validar o documento XML.

## 2.14 Login

Início de uma sessão, ou seja, a entrada do usuário no sistema.

## 2.15 Protocolo HTTP

*Hyper Text Transfer Protocol* - Protocolo de Transferência de Hipertexto (HTTP) é um protocolo da camada de "Aplicação" do modelo OSI, utilizado para transferência de dados na *World Wide Web*.

## 2.16 Protocolo TCP/IP

São protocolos de comunicação, o protocolo *Transmission Control Protocol* - Protocolo de Controle de Transmissão (TCP) e o *Internet Protocol* - Protocolo Internet (IP).

## 2.17 Servlet

*Servlet* é um programa que estende a funcionalidade de um *web server*, gerando conteúdo dinâmico e interagindo com os clientes, utilizando o modelo *request/response*.

## 2.18 Templates

São moldes, foram usados para gerar os artefatos (APÊNDICES).

## 2.19 UML

A *Unified Modeling Language* (UML) é uma linguagem de modelagem. Permite que desenvolvedores visualizem os produtos de seu trabalho em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significados, isto é, a semântica.