UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

DIEGO AGUIR SELZLEIN

INTEGRAÇÃO DAS TECNOLOGIAS EJB E GWT: UM ESTUDO EXPERIMENTAL PARA CONTROLE FINANCEIRO

MONOGRAFIA DE ESPECIALIZAÇÃO

MEDIANEIRA
2011

DIEGO AGUIR SELZLEIN

INTEGRAÇÃO DAS TECNOLOGIAS EJB E GWT: UM ESTUDO EXPERIMENTAL PARA CONTROLE FINANCEIRO

Monografia apresentada como requisito parcial à obtenção do título de Especialista na Pós Graduação em Engenharia de Software, da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Medianeira.

Orientador: Prof Me. Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Especialização em Engenharia de Software

TERMO DE APROVAÇÃO

Integração das tecnologias EJB e GWT: um estudo experimental para controle financeiro

Por

Diego Aguir Selzlein

Esta monografia foi apresentada às 11:00 h do dia 10 de dezembro de 2011 como requisito parcial para a obtenção do título de Especialista no curso de Especialização em Engenharia de Software, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. Os acadêmicos foram argüidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. M.Sc Fernando Schütz
UTFPR – Câmpus Medianeira
(orientador)

_____
Prof M.Sc. Alan Gavioli
UTFPR – Câmpus Medianeira

_____
Prof Dr. Claudio Leones Bazzi
UTFPR – Câmpus Medianeira

AGRADECIMENTOS

À Deus pelo dom da vida, pela fé e perseverança para vencer os obstáculos.

Aos meus pais, pela orientação, dedicação e incentivo nessa fase do curso de pós-graduação e durante toda minha vida.

Ao meu orientador professor Fernando Schütz, que me orientou, pela sua disponibilidade, interesse e receptividade com que me recebeu e pela prestabilidade com que me ajudou.

Agradeço aos pesquisadores e professores do curso de Especialização em Engenharia de Software, professores da UTFPR, Câmpus Medianeira.

Enfim, sou grato a todos que contribuíram de forma direta ou indireta para realização desta monografia.

RESUMO

SELZLEIN, Diego A. Integração das tecnologias EJB e GWT: um estudo experimental para controle financeiro. 2011. 42 p. Monografia (Especialização em Engenharia de Software). Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

Frameworks para desenvolvimento de sistemas computacionais auxiliam os desenvolvedores no processo de construção do aplicativo. Dois *frameworks* de desenvolvimento são focados neste trabalho. O primeiro é o Enterprise JavaBeans, ou EJB, que consiste em um conjunto de componentes voltados para o lado servidor de uma aplicação, disponibilizando serviços. O segundo é o Google Web Toolkit, ou GWT, que é voltado para o lado cliente do sistema e permite a criação de aplicações WEB ricas, que oferecem uma boa interface com o usuário. Este trabalho tem por objetivo desenvolver um estudo experimental para demonstrar a integração entre as tecnologias Enterprise JavaBeans e Google Web Toolkit. Além disso, apresenta a análise e o projeto da aplicação proposta, bem como as técnicas utilizadas para seu funcionamento.

Palavras-chave: Engenharia de Software para Web.

ABSTRACT

SELZLEIN, Diego A. An Integration of Technologies: EJB and GWT. 2011. 42 p. Monografia (Especialização em Engenharia de Software). Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

Frameworks for software development help developers in the software design process. Two development frameworks are the focus of this paper. The first one is the Enterprise JavaBeans, or EJB, that consists in a set of components to the server-side of an application, providing services. The second one is the Google Web Toolkit, or GWT, that is focused on the client-side of an application and allows to create rich Web applications, that offer a good user interface. This paper is focused on to develop an experimental study to demonstrate the integration of the technologies Enterprise JavaBeans and Google Web Toolkit. Moreover, presents the proposed application's analisys and design, and the technics used for its operation as well.

Keywords: Software Engineering for Web.

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AJAX	Assynchronous Javascript and XML
API	Application Programming Interface
DER	Diagrama Entidade Relacionamento
EE	Enterprise Edition
EJB	Enterprise JavaBeans
GWT	Google Web Toolkit
GWTP	Google Web Toolkit Platform
HTML	HyperText Markup Language
JNDI	Java Naming and Directory Interface
MDB	Message-driven bean
RIA	Rich Internet Application
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
TI	Tecnologia da Informação
UML	Unified Modeling Language
URL	Uniform Resource Locator
WODA	Write Once Deploy Anywhere
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Segundo Falbo (2005), desenvolvimento de sistemas é uma atividade de crescente importância. Ele ainda afirma que a utilização de computadores em áreas do conhecimento humano diversificadas gera crescente demanda por soluções computacionais. O uso da tecnologia tem importância nos processos de tomada de decisão de empresas que, cada vez mais, procuram agilidade e confiabilidade no processo (ANDRETO e AMARAL, 2006).

Os aplicativos computacionais na Web são uma excelente alternativa por permitirem a disponibilização e o processamento eficiente de uma grande quantidade de informação com uma diferença fundamental, que é a independência geográfica (ANDRETO e AMARAL, 2006).

Um *framework* pode ser definido como um *software* parcialmente completo projetado para ser instanciado (JUNIOR, 2006). Tal tecnologia define uma arquitetura para vários subsistemas, oferecendo recursos básicos para criá-los (JUNIOR, 2006).

O Google Web Toolkit, ou GWT, é um *framework* que permite ao desenvolvedor escrever e testar aplicações Web na linguagem de programação Java, que é convertida em JavaScript pelo *framework* para rodar no navegador do usuário (GUPTA, 2008).

O *framework* EJB é um padrão para desenvolvimento e publicação de componentes no lado servidor de uma aplicação (SRIGANESH, BROSE e SILVERMAN, 2006). Tal tecnologia permite e simplifica o desenvolvimento de aplicações distribuídas, transacionais, seguras e portáveis baseadas na tecnologia Java (ORACLE, 200-).

Assim sendo, o EJB seria capaz de disponibilizar a regra de negócio da aplicação, controlando suas conexões, ao passo que o GWT seria um ótimo cliente para consumir os serviços disponibilizados pelo primeiro.

1.1 OBJETIVO GERAL

Desenvolver um estudo experimental, utilizando a integração das tecnologias GWT e EJB, para controle financeiro pessoal.

1.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um estudo bibliográfico sobre Engenharia de Software na Web, Java Enterprise Edition, bem como as tecnologias GWT e EJB;
- Analisar e projetar a aplicação do estudo experimental para controle financeiro pessoal;
- Demonstrar, através de resultados de testes, a integração das tecnologias EJB e GWT.

1.3 JUSTIFICATIVA

As tecnologias EJB e GWT são robustas e oferecem diversas opções aos desenvolvedores de sistemas. Ambas são muito interessantes e usadas em inúmeros sistemas Web.

O GWT tem se mostrado uma ótima ferramenta na geração de interface do usuário final, pois oferece uma grande facilidade na internacionalização do sistema, na manipulação de arquivos de propriedades, nas parametrizações, na comunicação entre cliente e servidor, entre outros.

A tecnologia EJB é uma ótima ferramenta para disponibilizar serviços na Web pela sua robustez, alta escalabilidade, controle transacional e inúmeras funcionalidades oferecidas.

Assim sendo, essas tecnologias combinadas podem gerar uma aplicação robusta e com uma ótima experiência para o usuário, justificando um trabalho voltado às integração das mesmas.

2 REFERENCIAL BIBLIOGRÁFICO

Neste capítulo será feito um embasamento teórico, apresentando o resultado de pesquisas bibliográficas dos principais assuntos que serão abordados durante o trabalho.

2.1 ENGENHARIA DE *SOFTWARE*

Segundo Pressman (2006), um *software* pode ser definido como instruções “que quando executadas fornecem as características, função e desempenho desejados; estruturas de dados que permitem aos programas manipular adequadamente a informação”

O desenvolvimento de sistemas tem por objetivo resolver problemas e facilitar processos. Tais problemas e processos aumentam gradativamente de complexidade, exigindo maiores conhecimentos e habilidades (FALBO, 2005).

Segundo Falbo (2005), “chega-se a um ponto em que, dado o tamanho ou a complexidade do problema que se pretende resolver, (...) uma abordagem de engenharia é necessária”. Tal abordagem consiste em tratar o problema e decompô-lo em partes menores para então, resolvendo-se estas, seja alcançada a solução do todo.

A Engenharia de Software surgiu para melhorar a qualidade dos produtos de *software* e aumentar a produtividade no processo de desenvolvimento (FALBO, 2005).

Ela é “uma disciplina da engenharia que se ocupa de todos os aspectos da produção de *software*, desde os estágios iniciais da especificação do sistema até a manutenção desse sistema” (SOMMERVILLE, 2003). Segundo Sommerville (2003), engenheiros de *software* adotam uma abordagem sistemática e organizada em seu trabalho, uma vez que essa é a maneira mais eficaz de produzir *software* de alta qualidade.

2.1.1 Engenharia de Requisitos

Um requisito é um atributo necessário em um sistema, uma sentença que identifica uma capacidade, característica ou qualidade de um sistema a fim de que ele tenha utilidade para o usuário (YOUNG, 2004). Segundo Young (2004), requisitos são importantes por prover uma base para o trabalho de desenvolvimento que vem em seguida.

Requisitos são a base para qualquer projeto, definindo o que os *stakeholders* (usuários, fornecedores, desenvolvedores, empresas) em um novo sistema precisam dele (HULL, JACKSON e DICK, 2011). Para serem bem entendidos por todos, são geralmente escritos em linguagem natural e devem capturar a necessidade ou problema completamente e sem ambiguidade (HULL, JACKSON e DICK, 2011). Uma vez que os requisitos estão definidos, os desenvolvedores podem iniciar outros trabalhos técnicos, como projeto do sistema, desenvolvimento, teste, implementação e operação (YOUNG, 2004).

Stakeholders, segundo Hull, Jackson e Dick (2011), são indivíduos, grupos de pessoas, organizações ou outra entidade que tem interesse direto ou indireto no sistema e são as fontes de requisitos.

> A engenharia de requisitos ajuda os engenheiros de *software* a compreender melhor o problema que eles vão trabalhar para resolver. Ela inclui o conjunto de tarefas que levam a um entendimento de qual será o impacto do *software* sobre o negócio, do que o cliente quer e de como os usuários finais vão interagir com o *software*. (PRESSMAN, 2006).

Segundo Sommerville (2003), o estágio de engenharia de requisitos é particularmente importante, uma vez que os erros no mesmo inevitavelmente produzem problemas posteriores no projeto e na implementação do sistema.

2.1.2 Especificação dos Requisitos

Segundo Pressman (2006), no contexto de sistemas e *softwares*, o termo especificação coisas diferentes para pessoas diferentes. Para ele, “uma especificação pode ser um documento escrito, um modelo gráfico, um modelo

matemático formal, uma coleção de cenários de uso, um protótipo ou qualquer combinação desses elementos".

Especificação de requisitos "é a atividade de traduzir as informações coletadas durante a atividade de análise em um documento que defina um conjunto de requisitos" (SOMMERVILLE, 2003)

Para sistemas grandes, um documento escrito em linguagem natural combinado com modelos gráficos pode ser a melhor abordagem para representar seus requisitos. Por outro lado, cenários de uso podem ser o suficiente para sistemas menores que residam em ambientes técnicos bem entendidos (PRESSMAN, 2006).

O produto final de um engenheiro de requisitos é a especificação. Com ela é possível a realização das atividades seguintes. "Ela descreve a função e o desempenho de um sistema baseado em computador e as restrições que governarão seu desenvolvimento" (PRESSMAN, 2006).

2.1.3 Requisitos

De acordo com Sommerville (2003), os requisitos de sistema de *software* são, geralmente, classificados como:

1. Requisitos funcionais: declarações das funcionalidades que o sistema deve oferecer, como ele deve reagir e se comportar. Este tipo de requisito pode também declarar o que o sistema não deve fazer;
2. Requisitos não funcionais: restrições sobre os serviços ou as funções oferecidas pelo sistema. Destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros;
3. Requisitos de domínio: requisitos que se originam do domínio de aplicação do sistema e refletem características desse domínio. Podem ser requisitos funcionais ou não funcionais

2.1.3.1 Requisitos Funcionais

"Os requisitos funcionais para um sistema descrevem a funcionalidade ou os serviços que se espera que o sistema forneça" (SOMMERVILLE, 2003). Dois exemplos deste tipo de requisito poderiam ser:

1. Usuário deverá ser capaz de efetuar *login* no sistema.
2. Usuário deverá ser capaz de se cadastrar no sistema apenas interagindo com o mesmo, sem necessidade de intervenção de outra pessoa.

A especificação de requisitos funcionais de um sistema deve ser completa e consistente, ou seja, todas as funções requeridas pelo usuário devem estar definidas e os requisitos não devem apresentar definições contraditórias (SOMMERVILLE, 2003).

2.1.3.2 Requisitos Não Funcionais

"Os requisitos não funcionais, como o nome sugere, são aqueles que não dizem respeito diretamente às funções específicas fornecidas pelo sistema" (SOMMERVILLE, 2003). Conforme afirma Sommerville (2003), requisitos não funcionais podem definir restrições para o sistema; muitos deles dizem respeito ao sistema como um todo, e não a características individuais dele.

Entretanto, requisitos não funcionais nem sempre dizem respeito ao sistema de *software* a ser desenvolvido. Algumas vezes podem restringir o processo usado no desenvolvimento do mesmo (SOMMERVILLE, 2003).

2.1.3.3 Requisitos de Domínio

De acordo com Sommerville (2003), “são derivados domínio da aplicação do sistema, em vez de serem obtidos a partir das necessidades específicas dos usuários do sistema”. Podem ser requisitos funcionais em si, podem restringir os requisitos funcionais existentes ou estabelecer como devem ser realizados cálculos específicos (SOMMERVILLE, 2003).

Sommerville (2003) exemplifica um requisito de domínio para um sistema de biblioteca da seguinte forma:

- Em razão das restrições referentes a direitos autorais, alguns documentos devem ser excluídos imediatamente ao serem fornecidos. Dependendo dos requisitos dos usuários, esses documentos serão impressos localmente no servidor do sistema para serem encaminhados manualmente ao usuário ou direcionados para uma impressora da rede.

Esse requisito foi introduzido no sistema de biblioteca mencionado por Sommerville (2003) devido às leis de direitos autorais, que se aplicam aos materiais utilizados em bibliotecas.

2.1.4 Modelagem de Sistemas

Durante o desenvolvimento de um sistema, é comum o uso de figuras para auxiliar a visualização de alguns de seus aspectos (HULL, JACKSON e DICK, 2011). A modelagem de sistemas provê um meio de formalizar estas representações gráficas através de diagramas, não somente por definir uma sintaxe padrão como também por fornecer um meio de entendimento e comunicação de ideias associadas com o desenvolvimento (HULL, JACKSON e DICK, 2011).

Modelos são mais frequentemente representados visualmente e a informação é portanto representada através de diagramas conectados (HULL, JACKSON e DICK, 2011). Novos métodos como orientação a objetos possuem conceitos avançados de modelagem.

Para Hull, Jackson e Dick (2011), um bom modelo é aquele que é facilmente comunicado. Os modelos precisam ser usados para comunicação entre os membros da equipe de desenvolvimento e também com uma organização como um todo, incluindo os *stakeholders* (HULL, JACKSON e DICK, 2011).

Segundo Hull, Jackson e Dick (2011), uma modelagem pode ter inúmeros usos, como modelar as atividades de uma empresa inteira ou modelar um requisito funcional específico do sistema.

Ainda de acordo com Hull, Jackson e Dick (2011), modelagens podem trazer os seguintes benefícios:

- Encorajar o uso de um vocabulário precisamente definido e consistente no sistema;
- Permitir que projeções e especificações do sistema possam ser visualizadas em diagramas;
- Suportar a análise de sistemas através de uma disciplina definida;
- Permitir validação de alguns aspectos do projeto do sistema através de animações;
- Encorajar comunicação entre organizações diferentes através do uso de notações padrões comuns.

2.1.5 UML

A Linguagem de Modelagem Unificada (Unified Modeling Language – UML) foi desenvolvida com um propósito geral de padronizar notações que descrevem o comportamento estático e dinâmico de um sistema (YOUNG, 2004). Young (2004) afirma que ela é uma linguagem de modelagem visual que não intenciona ser uma linguagem de programação visual no sentido de possuir todo o suporte visual e semântico necessário para substituir linguagens de programação. Ainda segundo ele, a UML provê um modelo completo e formal para documentar um sistema.

A UML é uma forma eficaz de comunicação entre desenvolvedores, tornando possível também que um desenvolvedor, durante a implementação, possa retornar

aos modelos elaborados seguindo a modelagem UML e entender o que os analistas que os fizeram tinham em mente (CHONOLES e SCHARDT, 2003).

Segundo Chonoles e Schardt (2003), ela consiste numa linguagem de modelagem padronizada consistindo de um integrado conjunto de diagramas, compilado para auxiliar os desenvolvedores a cumprirem as tarefas:

- Especificação;
- Visualização;
- Projeção da arquitetura;
- Construção;
- Simulação e teste;
- Documentação.

Na Figura 1 pode ser visualizado um esquema com a maioria dos diagramas da UML. É possível que notações criadas para um diagrama UML específico sejam utilizadas em outro por algum motivo particular da aplicação que está sendo analisada. Isso da origem a um novo diagrama, que agrega notações de outro, e, por esse motivo, não há como afirmar com exatidão o número de diagramas da UML (CHONOLES e SCHARDT, 2003).

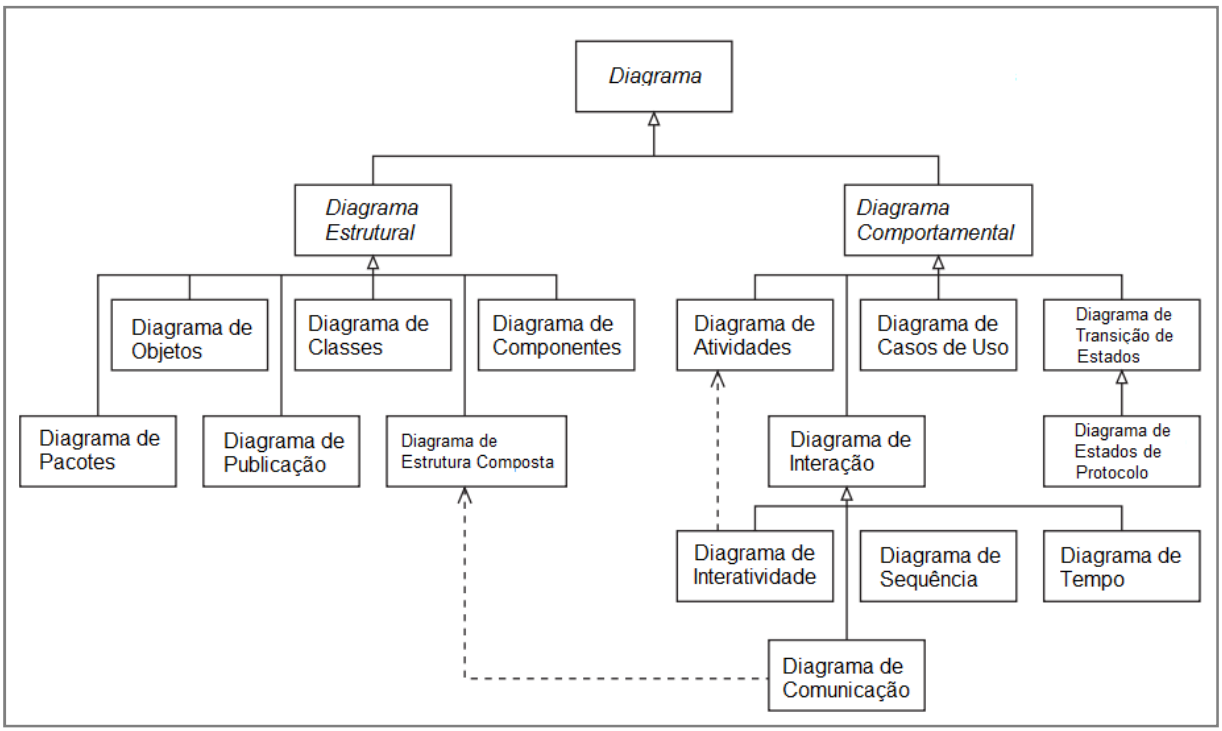

Figura 1 - Diagramas UML
Fonte: Adaptado de Chonoles e Schardt (2003)

Nem todos os diagramas representados na Figura 1 serão detalhados neste trabalho, tendo em vista que este não é o foco. Apenas aqueles que serão usados significativamente na modelagem do sistema proposto aqui serão discutidos.

2.1.5.1 Diagrama de Casos de Uso

Conforme são levantados os requisitos de um sistema, “uma visão geral das funções e características do sistema começam a se materializar” (PRESSMAN, 2006). Porém, avançar nas atividades mais técnicas de Engenharia de Software é complicado até que a equipe de *software* entenda como essas funções e características serão usadas por diferentes classes de usuários finais. Para alcançar tal entendimento, nas palavras de Pressman (2006), “desenvolvedores e usuários podem criar um conjunto de cenários que identifiquem uma linha de uso para o sistema a ser construído”. Tais cenários, conhecidos como casos de uso, descrevem como o sistema será utilizado.

Casos de uso definem a interação que toma lugar entre o sistema e o usuário (o ator) (HULL, JACKSON e DICK, 2011). O termo ator é usado para nomear uma entidade que reage com o sistema, podendo ser um tipo de usuário ou até outro sistema. Os diagramas de caso de uso contêm os atores e os casos de uso, além dos relacionamentos entre eles (HULL, JACKSON e DICK, 2011). Não há necessidade de atores serem humanos, pois, na verdade, eles representam papéis e cada um possuirá uma associação com no mínimo um caso de uso (HULL, JACKSON e DICK, 2011).

Os casos de uso se tornaram uma característica fundamenta da UML, que é uma linguagem de modelagem unificada, para descrever modelos de sistemas orientados a objetos (SOMMERVILLE, 2003).

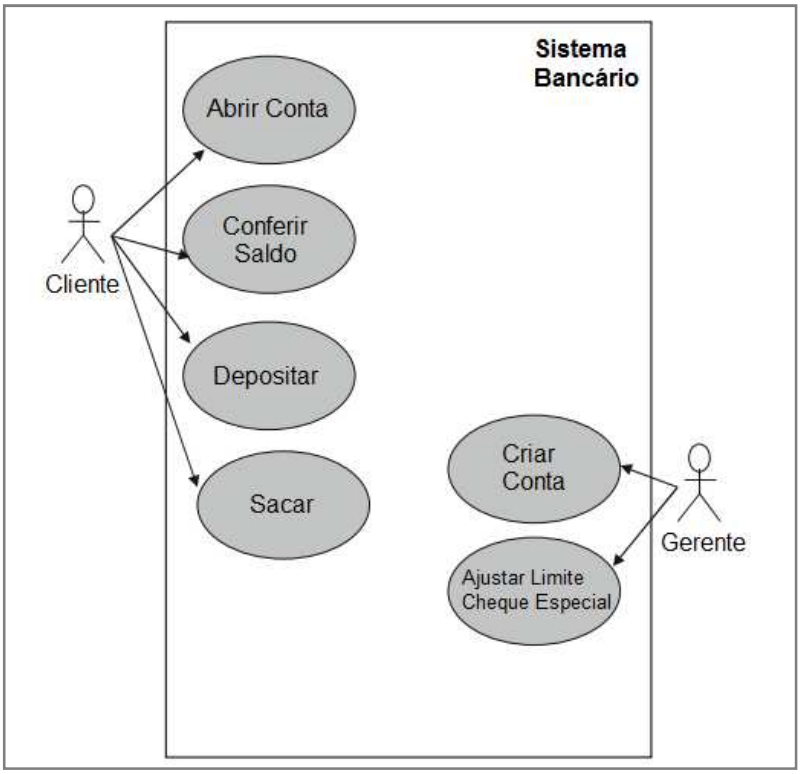

Figura 2 - Exemplo de diagrama de caso de uso
Fonte: Adaptado de Hull, Jackson e Dick (2011)

Como pode ser visualizado na Figura 2, o sistema é também definido no diagrama de casos de uso por um retângulo, com o nome do sistema exibido dentro dele (HULL, JACKSON e DICK, 2011). Informações textuais úteis normalmente são associadas com cada caso de uso.

2.1.5.2 Diagrama Entidade Relacionamento

A maioria dos sistemas utiliza bancos de dados para armazenar informações. Uma parte importante da modelagem de sistemas é definir o modo lógico dos dados processados pelo sistema (SOMMERVILLE, 2003). Diagramas Entidade Relacionamento (DERs) fornecem uma modelagem das entidades de interesse e os relacionamentos existentes entre elas (HULL, JACKSON e DICK, 2011).

Uma entidade é um objeto que pode ser identificado distinguidamente, assim como: consumidor, fornecedor e produto (HULL, JACKSON e DICK, 2011). Hull, Jackson e Dick (2011) afirmam ainda que uma propriedade (ou atributo) é a

informação que descreve a entidade. Segundo eles, um relacionamento entre entidades possui cardinalidade, que expressa a natureza da associação, podendo ser um para um, um para muitos ou muitos para muitos.

2.1.5.3 Diagrama de Classes

Programação orientada a objetos fornece uma abordagem de análise diferente da relativa à programação estruturada (HULL, JACKSON e DICK, 2011).

De acordo com Hull, Jackson e Dick (2011), o diagrama de classes é o diagrama básico da análise orientada a objetos. Um diagrama deste tipo apresenta classes que o sistema objeto orientado possuirá. Nele constam informações como:

- Atributos e seus tipos: para que a classe seja interessante e útil, ela deve possuir informações (propriedades) interessantes e úteis (CHONOLES e SCHARDT, 2003). Essas informações ou propriedades são chamadas de atributos. Depois de identificá-los é necessário definir qual o seu tipo. Um tipo especifica que tipo de dados determinado atributo poderá receber (CHONOLES e SCHARDT, 2003);
- Multiplicidade: permite identificar, principalmente, relações entre classes, como um para um, um para muitos ou muitos para muitos;
- Operações: representam os comportamentos que uma classe pode apresentar; é o conceito de método da orientação a objetos. Elas podem apresentar tipo de retorno e parâmetros, que, por sua vez, também apresentam tipos;
- Relacionamentos: ligações entre classes que indicam como uma se relaciona com outra, a multiplicidade entre elas, entre outras informações;
- Visibilidade: ao definir as classes, pode-se definir a visibilidade de seus atributos e operações, o que é visível e o que não é. Tipicamente, todos os atributos são privados, que significa que eles são vistos apenas dentro da própria classe (CHONOLES e SCHARDT, 2003).

Segundo Chonoles e Schardt (2003), cada atributo e cada operação deve possuir sua visibilidade definida.

Um exemplo de como atributos podem ser representados é apresentado no Quadro 1.

```
atributo : Tipo
```

Quadro 1 - Exemplo de atributo em um Diagrama de Classes
Fonte: Adaptado de Chonoles e Schardt (2003)

Formas de representação de multiplicidades podem ser visualizadas no Quadro 2.

Representação	**Significado**
1	Exatamente um
1..*	Um ou muitos
*	Muitos ou nenhum
0..1	Nenhum ou um

Quadro 2 - Representação de multiplicidades
Fonte: Adaptado de Chonoles e Schardt (2003)

No Quadro 3 é apresentada uma forma de representação de uma operação (método) em UML.

```
operação(argumento1: tipo, argumento2: tipo ...): tipo de retorno
```

Quadro 3 - Exemplo de representação de uma operação em UML
Fonte: Adaptado de Chonoles e Schardt (2003)

A UML exibe cada classe juntamente com seus atributos e comportamentos (operações). Cada tipo de informação (nome da classe, atributos e operações) possui seu próprio compartimento na representação (CHONOLES e SCHARDT, 2003). De acordo com Chonoles e Schardt (2003), seguindo os princípios de encapsulamento e ocultação de informações, os compartimentos podem não ser exibidos se desejado. Dessa forma, a representação padrão de uma classe fica conforme ilustrado na Figura 3.

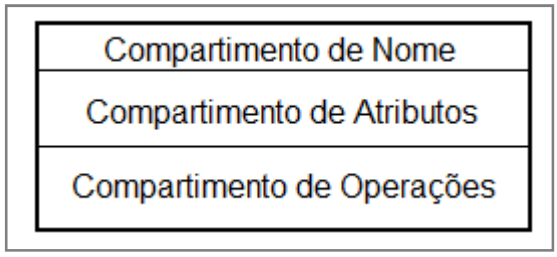

Figura 3 - Exemplo de representação de classe em UML
Fonte: Adaptado de Chonoles e Schardt (2003)

Na Tabela 1 estão explicadas as representações de visibilidade definidas pela UML.

Tabela 1 - Representação de visibilidades em UML

Símbolo	Visibilidade	Significado
+	Público	Qualquer objeto pode acessar.
-	Privado	Acessível apenas dentro da própria classe em que é declarado.
#	Protegido	Acessível apenas dentro da própria classe e pelos seus descendentes (classes que herdam).
~	Pacote	Apenas objetos dentro do mesmo pacote em que se encontra a classe que declara o atributo ou a operação podem acessar.

Fonte: Adaptado de Chonoles e Schardt (2003)

Na Figura 4 estão representadas duas classes. A classe `Conta` apresenta um atributo saldo e três operações, sendo elas `verificarSaldo`, `depositar` e `sacar`. A classe `Dono` apresenta apenas um atributo: nome. É possível entender, a partir deste diagrama, que a relação entre elas é de um para muitos, sendo que um dono pode ter muitas contas.

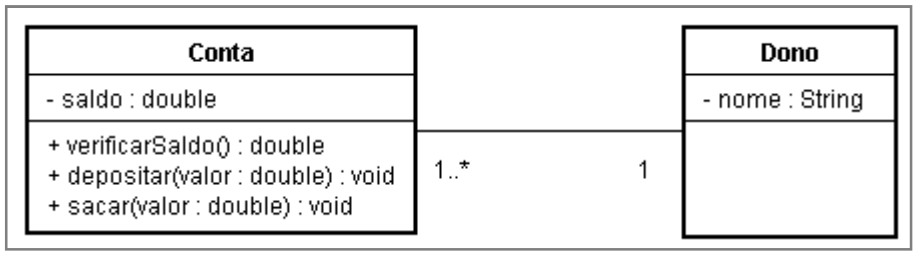

Figura 4 - Exemplo de diagrama de classes
Fonte: Adaptado de Hull, Jackson e Dick (2011)

2.1.6 Teste de Sistemas

Pressman (2006) traz algumas definições sobre testes de sistemas:

- Teste é um processo de execução de um programa com a finalidade de encontrar um erro;
- Um bom teste é aquele que tem alta probabilidade de encontrar um erro ainda não encontrado;
- Um teste bem sucedido é aquele que encontra um erro ainda não encontrado.

Segundo Sommerville (2007), um processo de teste de sistemas tem dois objetivos:

1. Demonstrar ao desenvolvedor e ao cliente que o sistema cumpre seus requisitos.
2. Descobrir falhas ou defeitos no sistema onde seu comportamento é incorreto, indesejável ou não corresponde à sua especificação.

Sommerville (2007) ainda afirma que o objetivo dos testes é convencer os desenvolvedores do sistema e os clientes que ele está bom o suficiente para ser usado.

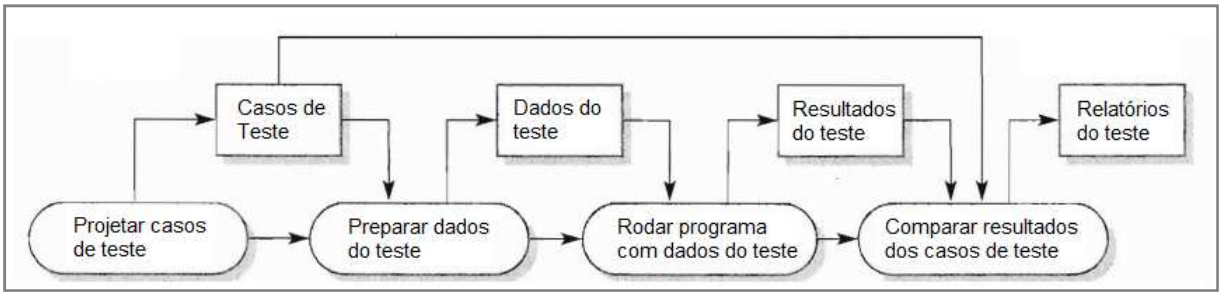

Figura 5 - Um modelo de processo de teste de *software*
Fonte: Adaptado de Sommerville (2007)

A Figura 5 apresenta um modelo de processo para testar um sistema. Nela se apresentam alguns novos conceitos, como:

- Casos de teste são especificações das entradas que o sistema receberá para o teste e a saída esperada do mesmo, juntamente com uma descrição do que está sendo testado (SOMMERVILLE, 2007).
- Dados do teste são as entradas concebidas para o teste (SOMMERVILLE, 2007). Tais dados podem ser gerados automaticamente, o que não ocorre com casos de teste, que não podem ser gerados automaticamente (SOMMERVILLE, 2007).

Pressman (2006) apresenta alguns princípios sobre testes:

- Princípio 1: Todos os testes devem estar relacionados com os requisitos dos clientes. O objetivo de um teste é descobrir erros. Do ponto de vista do cliente, os piores erros são aqueles que fazem com que o sistema não atenda aos seus requisitos.
- Princípio 2: Os testes devem ser planejados muito antes de serem executados. O planejamento dos testes pode começar assim que a análise do sistema for concluída. Todos os testes podem ser planejados antes que qualquer linha de código tenha sido gerada.
- Princípio 3: O princípio de Pareto se aplica aos testes de *software*. Relacionado com os testes, o princípio significa que 80% de todos os erros descobertos durante o teste estarão, provavelmente, relacionados a 20% de todos os componentes do sistema.
- Princípio 4: O teste deve iniciar no “varejo” e evoluir até o “atacado”. Os testes geralmente se concentram, inicialmente, em componentes individuais. Posteriormente, focam encontrr erros na integração entre componentes e, então, no sistema todo.
- Princípio 5: Testes exaustivos não são possíveis. A variedade de combinação de caminhos em um sistema, mesmo que de tamanho moderado, é excepcionalmente grande, ficando impossível testá-las todas. Testes exaustivos, onde todas as possibilidades de sequências de execuções do sistema são testadas, é impossível (SOMMERVILLE, 2007).

Testes não podem demonstrar que um sistema é livre de falhas ou que ele irá se comportar de forma específica em qualquer circunstância (SOMMERVILLE, 2007).

Uma estratégia de teste precisa ser baseada em um conjunto de possíveis casos de teste (SOMMERVILLE, 2007). Idealmente, companhias de desenvolvimento devem possuir políticas para escolher este conjunto ao invés de deixar tal trabalho para a equipe de desenvolvimento. Ainda conforme Sommerville (2007), tais políticas podem ser baseadas em políticas gerais de teste, como testar todas as instruções do programa ao menos uma vez. Também podem ser baseadas na experiência de uso do sistema e focar nas funcionalidades disponibilizadas pelo sistem. Sommerville (2007) exemplifica:

1. Todas as combinações que são acessadas através de menu devem ser testadas.
2. Combinações de funções que são acessadas através do mesmo menu devem ser testadas.
3. Onde uma entrada de dados é disponibilizada ao usuário, todas as funções precisam ser testadas tanto com valores válidos como para valores inválidos para a entrada.

2.2 *FRAMEWORKS* DE DESENVOLVIMENTO

Segundo Junior (2006), “a tecnologia de *frameworks* possibilita que uma família de produtos seja gerada a partir de uma única estrutura que captura conceitos mais gerais da família de aplicações”.

Existem diversas definições para *framework* na literatura. Para Junior (2006), “um *framework* é definido como um *software* parcialmente completo projetado para ser instanciado”. Ele afirma também que tal tecnologia define uma arquitetura para vários subsistemas, oferecendo recursos básicos para criá-los.

2.3 JAVA EE

Java EE é um conjunto robusto de serviços de *midleware* que facilitam o desenvolvimento do lado servidor das aplicações (SRIGANESH, BROSE e

SILVERMAN, 2006). “É a plataforma Java para criação de aplicações Web e aplicações corporativas distribuídas de larga escala (com uso de EJBs)” (LUZ, 2009).

Essa tecnologia possui segurança, performance, distribuição, concorrência, escalabilidade, controle de transações, persistência de dados dentre outros aspectos para criação de sistemas de qualidade (LUZ, 2009). Dessa forma, o desenvolvedor deve apenas se preocupar com as regras de negócio da aplicação, enquanto o Java EE cuida da infraestrutura. Tal aspecto permite criar aplicações independentes de plataforma e de servidor, o que se chama WODA, onde uma aplicação precisa ser escrita apenas uma vez e pode ser implantada em qualquer servidor compatível (LUZ, 2009).

2.4 *FRAMEWORK* EJB

Segundo Spinola (2006), aplicações empresariais são aquelas que, além de possuir funções empresariais, envolvem grandes quantidades de dados, simultaneamente acessados por diversos usuários. Segundo ele, tais aplicações dificilmente serão executadas isoladamente. Pelo contrário, geralmente cada uma consiste em uma parte do quebra-cabeça da paisagem de TI de uma companhia. Spinola ainda afirma que o desenvolvimento de tais *softwares* é um trabalho difícil por causa do desempenho, escalabilidade, concorrência e segurança que eles exigem.

Para Spinola (2006), a especificação do EJB que surgiu em março de 1998 veio em resposta a estas necessidades, tornando mais fácil a confecção de sistemas empresariais distribuídos e orientados a objeto.

Segundo Oracle (200-), Enterprise JavaBeans, também conhecido como EJB, é uma arquitetura de componentes para o lado servidor de uma aplicação. Afirma também que essa tecnologia permite e simplifica o desenvolvimento de aplicações distribuídas, transacionais, seguras e portáveis baseadas na tecnologia Java.

Para Luz (2009), “Enterprise JavaBeans (EJB) é a tecnologia Java padronizada para a criação de componentes em uma arquitetura distribuída, escalável, confiável e robusta”. Ele diz que utiliza-se EJBs para a camada de

negócios em aplicações de várias camadas. Segundo Luz (2009), para executar uma aplicação que utiliza tal tecnologia, é necessário um servidor de aplicação, chamado de *container*.

Enterprise Beans não são voltados para o lado cliente, são componentes para o lado servidor. São voltados para execução de operações para o lado servidor, que significa que devem estar disponíveis no maior período de tempo possível, além de serem tolerantes a falhas, transacionais, escaláveis e seguros (SRIGANESH, BROSE e SILVERMAN, 2006).

Através do EJB, é possível desenvolver e disponibilizar componentes distribuídos. Estes, também conhecidos como objetos distribuídos ou objetos remotos, são chamados pelo sistema remoto. Ou seja, podem ser chamados tanto de um processo cliente interno quanto por um sistema diferente na rede (SRIGANESH, BROSE e SILVERMAN, 2006).

De acordo com Sriganesh, Brose e Silverman (2006), os principais passos de um processo de chamada a um método remoto são:

1. O cliente chama um *stub*, que é um objeto do lado cliente que faz o papel de um proxy. Ele encapsula a comunicação com o servidor e sabe como fazer a chamada remota sobre a rede;
2. O *stub* faz uma chamada através da rede para um *skeleton*, que é um objeto do tipo proxy do lado servidor e também faz o papel de encapsular a comunicação com o cliente através da rede. O *skeleton* entende como receber os dados e entender os parâmetros enviados pelo cliente;
3. O *skeleton* delega a chamada para o objeto implementado apropriado. Este objeto atende a chamada e faz seu trabalho, retornando, então, ao *skeleton*, que retorna ao *stub*, o qual, finalmente, retorna ao cliente.

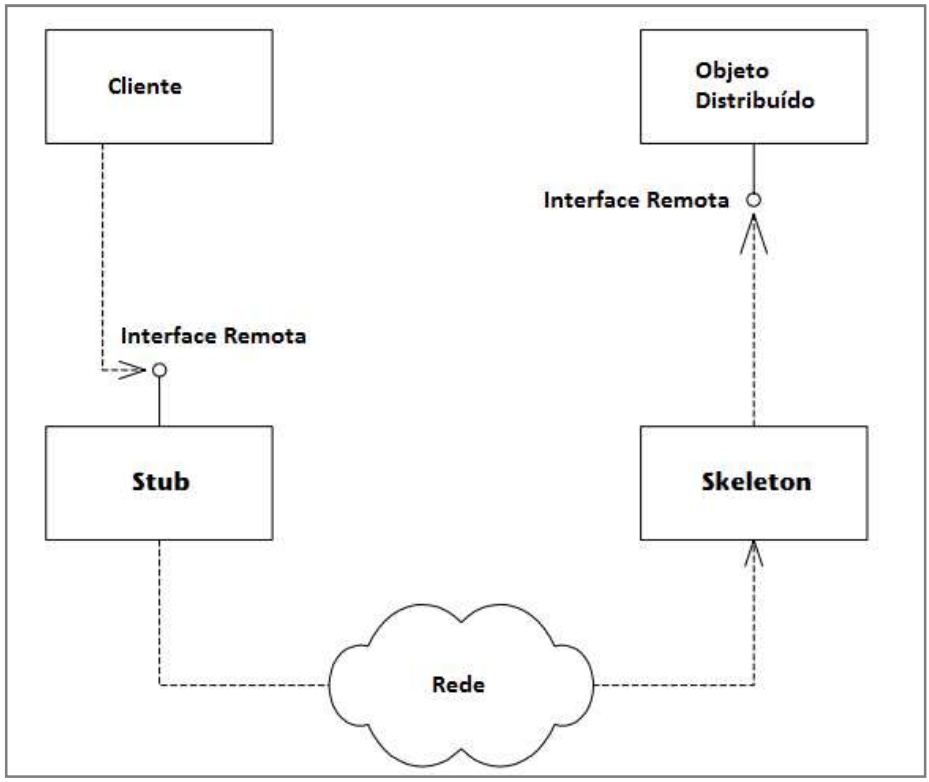

Figura 6 - Invocação de método remoto
Fonte: Adaptado de Sriganesh, Brose e Silverman (2006)

Quando um cliente invoca uma instância de um objeto EJB remoto, ele nunca invoca métodos diretamente na verdadeira instância do objeto. Tal invocação é interceptada pelo *container* EJB, que a delega para a instância. Fazendo isso, o *container* EJB pode controlar o acesso às instâncias e quando uma nova instância deve ou não deve ser criada (SRIGANESH, BROSE e SILVERMAN, 2006).

2.4.1 Enterprise Beans

Um Enterprise Bean é um componente de *software* do lado servidor que pode ser distribuído em um ambiente multi-camadas. Ele é escrito usando a API EJB e é posto em funcionamento dentro de um *container* EJB, que o disponibiliza com uma gerência de ciclo de vida, segurança, controle transacional, entre outros (SRIGANESH, BROSE e SILVERMAN, 2006).

O cliente que utilizará o *bean* pode ser qualquer coisa: um Servlet, uma aplicação *standalone*, um applet ou, inclusive, outro EJB (SRIGANESH, BROSE e SILVERMAN, 2006).

De acordo com Sriganesh, Brose e Silverman (2006), Enterprise JavaBeans são categorizados em vários tipos. Dependendo dos requisitos deles, é possível escolher um tipo de *bean* que melhor se adequa:

- **Session Beans** (*beans* de sessão): modelam processos de negócio, executam ações. Podem executar, por exemplo, acessos a um banco de dados, chamar outro sistema ou mesmo outro EJB. São subdivididos em dois tipos: *stateless* e *stateful*. O primeiro não mantém seu estado entre múltiplas chamadas de usuário, enquanto o segundo mantém.
- **Message-driven Beans** (*beans* dirigidos a mensagens): também conhecidos como MDB's, são semelhantes aos *beans* de sessão, pois executam ações. São diferentes pois podem ser chamados unicamente com o recebimento implícito de mensagens. Não há maneira de invocar um método em um MDB diretamente.
- **Entity Beans** (*beans* entidade): modelam dados de negócio. Podem ser entendidos como substantivos porque são objetos de dados, carregam informação do banco de dados.

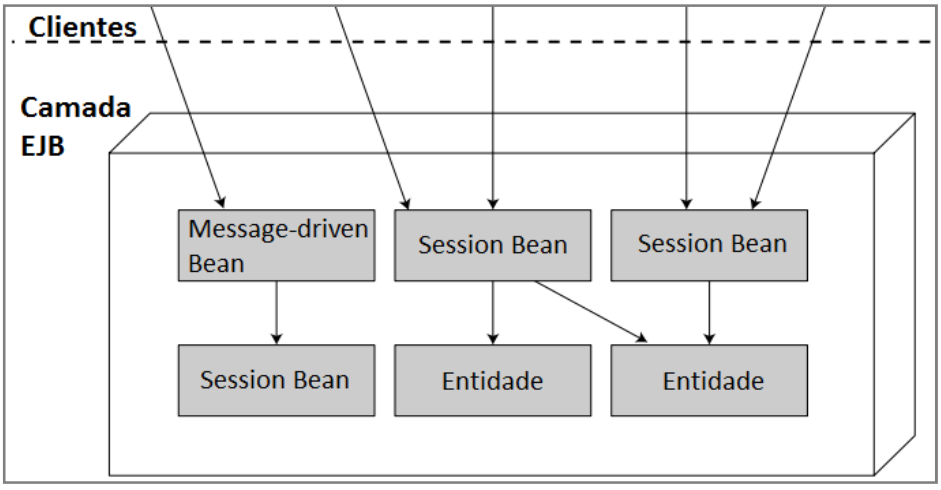

Figura 7 - Sistema EJB acessado por vários clientes
Fonte: Adaptado de Sriganesh, Brose e Silverman (2006)

2.4.2 O Uso de Anotações em EJB

A especificação EJB define várias anotações java úteis para o desenvolvimento de *beans*. Isso ajuda significantemente a programação de um EJB. Através de anotações, é possível definir metadados do *bean* juntamente com seu próprio código (SRIGANESH, BROSE e SILVERMAN, 2006). No Quadro 4 pode ser visualizado um exemplo de *bean* do tipo *stateful* apresentando um método e duas anotações.

```
@Stateful
public class ExampleBean implements BeanBusinessInterface {

	@Remove
	public void removeBean() {
		// Fechar qualquer recurso que foi aberto para
		// atender as requisições.
	}

}
```

Quadro 4 - Exemplo de *bean*
Fonte: Adaptado de Sriganesh, Brose e Silverman (2006)

Conforme afirmam Sriganesh, Brose e Silverman (2006), a anotação `@Stateful` indica um *bean* de sessão do tipo *stateful*. A anotação `@Remove`, por sua vez, indica que o método a que se refere deve ser chamado pelo *container* no momento em que o mesmo irá destruir a instância.

De acordo com o que é visível no Quadro 4, o desenvolvimento do EJB é simples. Um desenvolvedor pode anotar seu código e esperar que os compiladores, geradores de código, ferramentas de *deploy* ou o que quer que estiver trabalhando para disponibilizar o EJB, cuidem das semânticas apropriadas.

2.4.3 Exemplo

Para melhor entendimento, será feito uso de um exemplo de criação de um *bean* e seu acesso pelo cliente. O exemplo trata-se de um *bean* de sessão do tipo

stateless que apresenta apenas um método `alo()`. Quando este método é invocado, ele retorna uma saudação.

Primeiramente é criada uma interface que modele o *bean*, como a visualizada no Quadro 5.

```
package ejb.exemplo;

public interface Alo {
	public String alo();
}
```

Quadro 5 - Exemplo de interface que modela um *bean*
Fonte: Adaptado de Sriganesh, Brose e Silverman (2006)

A classe `AloBean` será a implementação do *bean* que, além de implementar o método alo, possuirá algumas anotações de configuração, como apresentado no Quadro 6.

```
package ejb.exemplo;

import javax.ejb.Remote;
import javax.ejb.Stateless;

@Stateless
@Remote(Alo.class)
public class AloBean implements Alo {

	@Override
	public String alo() {
		return "Olá";
	}
}
```

Quadro 6 - Exemplo de implementação do *bean*
Fonte: Adaptado de Sriganesh, Brose e Silverman (2006)

Nota-se que são usadas duas anotações na classe `AloBean`. A anotação `@Stateless` configura o *bean* como *bean* de sessão do tipo *stateless*, enquanto a anotação `@Remote` permite a configuração do suporte do *bean* para o cliente remoto através da interface Alo (SRIGANESH, BROSE e SILVERMAN, 2006). No lugar de um descritor de *deploy* estão sendo usadas anotações.

Para que o serviço disponibilizado seja acessado por uma aplicação cliente Java e o método `alo()` seja invocado, pode-se seguir o exemplo no Quadro 7.

```
package cliente;

import javax.naming.Context;
import javax.naming.InitialContext;

import br.com.financero.ejb.especifico.Alo;

/**
 * Esta classe é um exemplo de código de um cliente
 * que invoca um método em um simples bean de sessão remoto.
 */
public class AloCliente {
      public static void main(String[] args) throws Exception {

            /**
             * Obtém um contexto JNDI
             *
             * Este contexto inicial é um ponto incial para
             * conectar à uma árvore JNDI.
             */
            Context ctx = new InitialContext();
            Alo alo = (Alo) ctx.lookup("ejb.exemplo.Alo");

            /**
             * Chama o método disponibilizado.
             */
            System.out.println(alo.alo());
      }
}
```

Quadro 7 - Exemplo de acesso ao *bean* a partir de um cliente em Java
Fonte: Adaptado de Sriganesh, Brose e Silverman (2006)

Percebe-se que, para o cliente conseguir um objeto de acesso ao EJB, ele precisa adquirir um contexto inicial JNDI e localizar a referência para a interface do *bean* através deste contexto. JNDI significa Java Naming and Directory Interface e permite a busca de uma referência através de um nome. O *framework* EJB permite que os *beans* sejam encontrados através de uma chamada utilizando JNDI.

2.4.4 *Container* EJB

Um *container* EJB é responsável pelo controle dos EJBs (os *enterprise beans*). Segundo Sriganesh, Brose e Silverman (2006), a maior responsabilidade dele é prover um ambiente seguro, transacional e distribuído onde os EJBs podem ser executados. As informações relativas aos EJBs que serão disponibilizados pelo *container*, os metadados, que carregam a informação necessária para que o

container disponibilize cada *bean*, pode ser dada através de um arquivo XML chamado descritor de *deployment* ou através de anotações java (SRIGANESH, BROSE e SILVERMAN, 2006).

2.5 SERVIDOR DE APLICAÇÃO GLASSFISH

Em 2005, a empresa Sun criou o projeto Glassfish, tendo como seu objetivo o desenvolvimento de um servidor de aplicações totalmente certificado em Java EE (GONCALVES, 2009). Em seu núcleo, na parte do *container* Web, ele possui muitas heranças do servidor de aplicações Tomcat, sendo que uma aplicação feita para ser executada no servidor Tomcat possivelmente funcionará de igual maneira no servidor Glassfish (GONCALVES, 2009).

Segundo Luz (2009), “o Glassfish é um servidor de aplicações Java EE maduro, robusto e livre”. Ele ainda afirma que se trata de uma “implementação de referência para a especificação do EJB 3.0, e agora do 3.1”.

A aplicação EJB desenvolvida neste trabalho será construída voltada para funcionar neste servidor de aplicação.

2.6 *FRAMEWORK* DE DESENVOLVIMENTO WEB GWT

O Google Web Toolkit (GWT) é um conjunto de ferramentas para desenvolver aplicações Web em AJAX com Java (VOGEL, 2011). Segundo Smeets, Uri e Bankras (2009), ele “é um *framework* de código aberto, que facilita a criação de RIAs para desenvolvedores Java”. “Permite o desenvolvimento de aplicações JavaScript em Java e isso é obtido através da parte mais importante do GWT, o compilador de Java para JavaScript” (SMEETS, URI e BANKRAS, 2009).

Aplicações GWT são descritas como módulos. Uma aplicação com o nome de “teste”, por exemplo, é descrita pelo arquivo de configuração “teste.gwt.xml”. Um *entry point* (ponto de entrada) em uma aplicação GWT define um ponto inicial da mesma, semelhante ao método `main(String[] args)` de um programa Java.

Um módulo GWT define um ou mais *entry points*. A classe Java que define um *entry point* precisa implementar a interface `com.google.gwt.core.client.EntryPoint`, que a obriga a implementar o método `onModuleLoad()` (VOGEL, 2011).

Segundo Vogel (2011), um módulo é conectado a uma página HTML, chamada de "*host page*" (página hospedeira). O código de uma aplicação Web GWT executa dentro desta página.

2.6.1 Um Sistema em GWT

O esqueleto básico da aplicação GWT consiste em uma página HTML hospedeira e um *entry point*.

Como a criação de componentes e outros itens de visualização é feita dentro do *entry point*, em Java, é bom haver um componente na página hospedeira que receberá os elementos criados. Um exemplo pode ser visualizado no Quadro 8.

```
...
<body>
      <table align="center">
            <tr>
                  <td id="slot1"></td>
            </tr>
      </table>
</body>

...
```

Quadro 8 - Exemplo de página hospedeira
Fonte: Adaptado de Smeets, Uri e Bankras (2009)

O *framework* GWT oferece uma forma de se obter uma referência a um elemento da página hospedeira como um objeto. Neste podem ser adicionados outros elementos. Dessa forma, é possível que o elemento com ID equivalente a "slot1" seja obtido e nele sejam adicionados os elementos que se deseja.

O GWT fornece diversos componentes visuais prontos para serem adicionados à página hospedeira. No pacote `com.google.gwt.user.client.ui` podem ser encontrados: `Label`, `TextBox`, `Button`, `PasswordTextBox` entre outros. Cada um equivale a um elemento HTML.

No Quadro 9 é possível visualizar um exemplo de implementação do método `onModuleLoad()` que carrega alguns dos componentes citados no elemento com ID igual a “slot1”. O resultado obtido daquele código pode ser visualizado na Figura 8.

```
	public void onModuleLoad() {
		final TextBox username = new TextBox();
		final PasswordTextBox password = new PasswordTextBox();
		final Button button = new Button("Logon");
		final VerticalPanel panel = new VerticalPanel();
		panel.add(new Label("username"));
		panel.add(username);
		panel.add(new Label("password"));
		panel.add(password);
		panel.add(button);
		RootPanel.get("slot1").add(panel);
	}
```

Quadro 9 - Exemplo de onModuleLoad()
Fonte: Smeets, Uri e Bankras (2009)

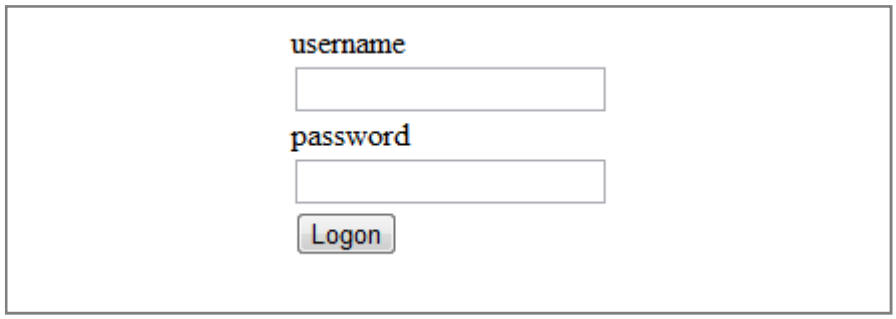

Figura 8 - Exemplo de visualização gerada pelo GWT

2.6.2 O Lado Servidor do GWT

Uma aplicação Web, para não ser estática, precisa persistir seus dados (SMEETS, URI e BANKRAS, 2009). Para isso, o GWT fornece um lado servidor que tem o poder de executar tal serviço e ele o faz de forma simples.

Um método que ele fornece para tanto se chama GWT RPC. Com ele, praticamente todos os detalhes de comunicação são abstraídos, sendo que o desenvolvedor precisa somente especificar uma interface com os métodos que serão invocados no lado servidor (SMEETS, URI e BANKRAS, 2009). Os serviços

disponibilizados não possuem limitação de métodos (SMEETS, URI e BANKRAS, 2009).

O GWT RPC utiliza o recurso de Servlets, introduzido com a especificação Java EE. “Um Servlet é uma extensão de um Web *container*, está habilitado a receber pedidos e formular uma resposta aplicável para eles” (SMEETS, URI e BANKRAS, 2009). Um exemplo de interface que define como um serviço irá se comportar pode ser visto no Quadro 10.

```
@RemoteServiceRelativePath("loginService")
public interface LoginService extends RemoteService {
      public boolean login(String usuario, String senha);
}
```

Quadro 10 - Exemplo de interface para um serviço remoto do GWT

Tal interface vai ser acessada pelo lado cliente. Portanto, ela precisa estar disponível para ele, ou seja, precisa estar dentro do pacote cliente da aplicação, que é aquele onde se encontra o *entry point*. Toda interface que define um serviço deve extender a interface `com.google.gwt.user.client.rpc.RemoteService` para que o compilador GWT entenda que ela define uma interface RPC (SMEETS, URI e BANKRAS, 2009). A anotação `@RemoteServiceRelativePath` define o caminho para chegar a este serviço.

No Quadro 11 encontra-se um exemplo de implementação da interface de serviço. Essa classe deve ser criada em um outro pacote, que não o pacote cliente. Ela extende a classe `com.google.gwt.user.server.rpc.RemoteServiceServlet`, que, segundo Smeets, Uri e Bankras (2009), adiciona outra camada de abstração na classe `Servlet` e abstrai a serialização, além de determinar o método que precisa ser invocado e invocá-lo.

```
public class LoginServiceImpl extends RemoteServiceServlet implements
LoginService {
	private static final long serialVersionUID = 35184378472223326L;

	@Override
	public boolean login(String usuario, String senha) {
		if (usuario.equals("admin") && senha.equals("1234"))
			return true;
		return false;
	}
}
```

Quadro 11 - Exemplo de implementação de um serviço GWT RPC

Como o GWT RPC é assíncrono, é necessário definir uma versão assíncrona da interface que define o serviço. Para isto, é preciso outra interface como a exibida no Quadro 12.

```
public interface LoginServiceAsync {

	void login(String usuario, String senha, AsyncCallback<Boolean>
callback);
}
```

Quadro 12 - Exemplo de interface assíncrona para o serviço em GWT RPC

Como pode ser visualizado, tal interface define os mesmos métodos da interface síncrona. A diferença é que na assíncrona eles não possuem retorno. Além disso, eles recebem um parâmetro a mais: uma instância da interface `com.google.gwt.user.client.rpc.AsyncCallback`, que, através de *generics*, especifica o tipo do retorno.

Para disponibilizar o serviço para acesso, é necessário definir o Servlet no arquivo <raíz do projeto>\war\WEB-INF\web.xml da forma como é mostrado no Quadro 13.

```
  <servlet>
    <servlet-name>loginServlet</servlet-name>
    <servlet-class>br.com.exemplotcc.server.LoginServiceImpl</servlet-
class>
  </servlet>
  <servlet-mapping>
    <servlet-name>loginServlet</servlet-name>
    <url-pattern>/exemplotcc/loginService</url-pattern>
  </servlet-mapping>
```

Quadro 13 - Exemplo de definição de servlet no arquivo web.xml

Como é apresentado, deve haver uma definição de um Servlet, possuindo ele um nome e a definição da classe que o implementa. Também faz-se necessário mapeá-lo através da *tag* “servlet-mapping”. Nela é informado seu nome e a URL (caminho) que ele possui na aplicação. A URL deve ser o nome da aplicação seguido do nome do serviço, que foi definido na anotação `@RemoteServiceRelativePath` da interface síncrona.

Uma forma de invocar o método exemplificado pode ser visualizada no Quadro 14. O método `com.google.gwt.core.client.GWT.create(Class)` tem diversas funções. Uma delas é gerar a implementação da interface assíncrona dos serviços. Isso ocorre pois o GWT foi feito para gerar código específico para o navegador que o usuário estiver usando. Assim, se o usuário estiver usando o Firefox, receberá código específico para ele, enquanto um usuário utilizando o Internet Explorer receberá código específico para este (SMEETS, URI e BANKRAS, 2009).

```
LoginServiceAsync s = (LoginServiceAsync) GWT.create(LoginService.class);
s.login("admin", "1234", new AsyncCallback<Boolean>() {

	@Override
	public void onSuccess(Boolean result) {
		if (result)
			System.out.println("Login obteve êxito.");
		else
			System.out.println("Usuário ou senha inválidos.");
	}

	@Override
	public void onFailure(Throwable caught) {
		caught.printStackTrace();
	}
});
```

Quadro 14 - Exemplo de invocação de um serviço remoto em GWT RPC

Ainda no Quadro 14 é visível que o método recebe os dois parâmetros necessários além de uma instância de `AsyncCallback`, na qual será chamado o método `onSuccess(T result)` em caso de êxito e o método `onFailure(Throwable caught)` em caso de exceção. Essa chamada será assíncrona. Ou seja, a execução não será interrompida na linha que o método apresentado é chamado até o fim de sua execução.

2.6.3 Internacionalização

> Quando se desenvolve qualquer tipo de aplicação, um dos mais importantes fatores que se deve ter em mente é o seu usuário alvo. A importância do usuário aumenta ainda mais quando se desenvolve RIAs, pois elas, potencialmente, estão abertas a um vasto número de usuários em todo o globo. (...) O RIA precisa ser escrito de tal maneira que sua lógica de representação (...) pode se adaptar ao país e ao idioma de seus usuários. (SMEETS, URI e BANKRAS, 2009).

Segundo Smeets, Uri e Bankras (2009), o *framework* GWT, assim como inúmeros outros, oferece suporte à internacionalização, tornando a aplicação adaptável a diferentes idiomas. Para tal, o *framework* em questão faz uso do conceito de *resource bundles* do Java.

"Um *resource bundle* em Java é uma abstração da procura de mensagens de textos localizadas, e é o mecanismo padrão que aplicações Java usam para suporte ao i18n" (SMEETS, URI e BANKRAS, 2009). A palavra "i18n" é a abreviação de "*internacionalization*" (internacionalização), onde as duas letras (*i* e *n*) se referem aos caracteres iniciais e finais da palavra e o número 18 representa os 18 medianos(SMEETS, URI e BANKRAS, 2009). Segundo Smeets, Uri e Bankras (2009), a classe `PropertyResourceBundle` é a implementação padrão e mais comum de *resource bundles* e é baseada em arquivos de propriedades. Estes arquivos associam chaves com mensagens e, através de vários deles, é possível configurar mensagens diferentes para diversos idiomas.

Assim, a internacionalização em GWT pode ser feita através do uso desses, sendo que eles devem possuir as mesmas chaves mapeadas para as mensagens no idioma a que o arquivo se refere. Um exemplo pode ser visto no Quadro 15 e no Quadro 16, onde estão exibidas as mensagens para os idiomas português do Brasil e inglês americano respectivamente.

```
campoContaDescricao=Descrição
campoMovimentacaoCategoria=Categoria
campoMovimentacaoConta=Conta
campoMovimentacaoData=Data
campoMovimentacaoDescricao=Descrição
campoMovimentacaoHistorico=Histórico
campoMovimentacaoValor=Valor
campoUsuarioEmail=E-mail
campoUsuarioLogin=Login
campoUsuarioNome=Nome
campoUsuarioSenha=Senha
campoUsuarioSituacao=Situação
```

Quadro 15 - Arquivo de mensagens em português do Brasil (messages_pt_BR.properties)

```
campoContaDescricao=Description
campoMovimentacaoCategoria=Category
campoMovimentacaoConta=Account
campoMovimentacaoData=Date
campoMovimentacaoDescricao=Description
campoMovimentacaoHistorico=History
campoMovimentacaoValor=Value
campoUsuarioEmail=E-mail
campoUsuarioLogin=Login
campoUsuarioNome=Name
campoUsuarioSenha=Password
campoUsuarioSituacao=Situation
```

Quadro 16 - Arquivo de mensagens em inglês dos Estados Unidos (messages_en_US.properties)

Existe uma prioridade na hora de escolher o arquivo de propriedades que será utilizado pelo GWT. Primeiro será buscado um arquivo que termine como o idioma mais *underscore* mais o país do usuário. Se não encontrado, será buscado um arquivo terminando com o idioma do mesmo. Caso este também não seja encontrado, será buscado um que não tenha nenhuma terminação indicando idioma ou local, podendo ser ele “messages.properties”, por exemplo (SMEETS, URI e BANKRAS, 2009).

2.6.4 A Interface Constants

O uso dos arquivos de *resource bundles* pode ser feito através da criação de uma interface que extenda `com.google.gwt.i18n.client.Constants`. Nela o desenvolvedor precisa declarar um método para cada mensagem mapeada nos arquivos, cujo nome deve corresponder a chave pela qual a mensagem foi

mapeada. Para utilizar as mensagens definidas no Quadro 15, por exemplo, a interface precisa declarar os métodos como demonstrado no Quadro 17.

```
public interface MinhasMensagens extends Constants {
      public String campoContaDescricao();
      public String campoMovimentacaoCategoria();
      public String campoMovimentacaoConta();
      public String campoMovimentacaoData();
      public String campoMovimentacaoDescricao();
      public String campoMovimentacaoHistorico();
      public String campoMovimentacaoValor();
      public String campoUsuarioEmail();
      public String campoUsuarioLogin();
      public String campoUsuarioNome();
      public String campoUsuarioSenha();
      public String campoUsuarioSituacao();
}
```

Quadro 17 - Interface de exemplo para internacionalização

"No GWT (...) os arquivos de propriedades devem estar no mesmo pacote próximo à interface que herdou `Constants`" (SMEETS, URI e BANKRAS, 2009). Para utilizar o que foi feito, basta usar o método `GWT.create(Class)`, parametrizando a classe da interface que herdou `Constants` (no exemplo, `MinhasMensagens.class`). O GWT se encarregará de criar um objeto implementando a interface de modo que cada método retorne a mensagem a qual corresponde.

2.6.5 A Interface Messages

Há momentos em que as mensagens variam dependendo do contexto. Ou seja, a mensagem precisa receber uma espécie de parâmetro, pois irá variar em determinados momentos. Para isto, o GWT fornece a interface `com.google.gwt.i18n.client.Messages`. No Quadro 18, é possível ver uma mensagem incompleta. Ela tem a finalidade de receber um parâmetro que será variável.

```
campoConfirma=Redigite {0}
```

Quadro 18 - Mensagem dependente de um contexto

Para isto funcionar, a interface que define as mensagens deve herdar `Messages` ao invés de `Constants`. “A principal diferença é que, quando se está usando `Messages`, os métodos com argumentos e as mensagens localizadas mapeadas podem definir posições especiais onde esses argumentos ficarão” (SMEETS, URI e BANKRAS, 2009). O método correspondente ao exemplo seria como demonstrado no Quadro 19.

```
public String campoConfirma(String campo);
```

Quadro 19 - Método dependente de um contexto

A criação do objeto que herda a interface e trata essas mensagens é feita da mesma forma: através do `GWT.create(Class)`. Dessa forma, o GWT entende que o primeiro parâmetro deverá ocupar o local onde está `{0}`, enquanto o segundo deve estar onde está `{1}` e assim sucessivamente.

3 PROCEDIMENTOS METODOLÓGICOS

O estudo apresentado por este trabalho será enriquecido com o projeto e desenvolvimento de uma aplicação, como estudo experimental que utilizará as tecnologias discutidas na revisão bibliográfica. Tal *software* apresentará funções para o controle financeiro pessoal de seus usuários apenas com finalidade de exemplificação.

3.1 OBJETIVO DO SISTEMA

O sistema de controle financeiro desenvolvido tem por objetivo organizar de forma fácil as movimentações financeiras pessoais dos usuários. Tem finalidade de permitir o controle das movimentações de diversas contas do usuário, como conta pessoal, conta em banco, ou outros, à definição do usuário, separando cada lançamento de movimentação por categoria e histórico. Esse último tem por finalidade definir se cada movimento a ele relacionado se trata de uma entrada ou saída. O sistema não é voltado para controle empresarial.

3.2 REQUISITOS FUNCIONAIS

A aplicação compreende os seguintes requisitos funcionais:

- **RF01**: Registrar movimentos de valores, tanto entradas como saídas;
- **RF02**: Registrar categorias, que podem ser usadas para categorizar as movimentações, sendo elas, por exemplo, “Alimentação”, “Transporte” ou “Saúde”. Dessa forma o usuário pode definir qual é o tipo da movimentação;
- **RF03**: Registrar contas, que são usadas no lançamento de movimentações;

- **RF04**: Registrar históricos, que são usados no lançamento de movimentações para definir se a movimentação se refere a uma entrada ou saída;
- **RF05**: Separar históricos, contas, categorias e movimentações por usuários. Um usuário não tem acesso as informações de outro usuário;
- **RF06**: Oferecer opção de o usuário se cadastrar, sendo que o e-mail informado pelo usuário deve ser confirmado válido, a partir de envio de uma mensagem para o mesmo com um link de confirmação;
- **RF07**: Permitir que usuário efetue *login*. O *login* deve ser permitido apenas depois que o usuário confirmou seu e-mail através do *link* de confirmação mencionado no RF06;
- **RF08**: Listar as últimas 50 movimentações inseridas pelo usuário na tela inicial, exibida após *login* efetuado com sucesso.

3.3 REQUISITOS NÃO FUNCIONAIS

O sistema ainda apresenta os seguintes requisitos não funcionais:

- **RNF01**: A disponibilização do sistema em nível de servidor deve ser feita em EJB, utilizando a linguagem de programação Java;
- **RNF02**: Deve haver um Enterprise JavaBean disponibilizado para cada entidade que será acessada pela camada cliente, sendo elas categoria, usuário, conta, movimentação e histórico;
- **RNF03**: A camada cliente deve ser representada por uma aplicação GWT, desenvolvida utilizando a linguagem de programação Java, que apenas faz chamadas ao servidor e exibe resultados para o usuário;
- **RNF04**: O servidor em EJB deverá dispor todas as funcionalidades do sistema apenas para a rede onde se encontra o cliente GWT;
- **RNF05**: Cada usuário deve possuir acesso apenas aos dados cadastrados por ele mesmo, sendo que ele não pode alterar, incluir ou visualizar dados de outros usuários;

- **RNF06**: Cada método disponibilizado pela aplicação EJB que trabalhe com dados que são separados por usuários, como categorias, históricos, contas e movimentos, deve receber por parâmetro o usuário referente aos dados que estão sendo alterados, incluídos ou consultados. Esse usuário parametrizado deve ser validado, através da verificação da corretude do *login* e da senha, antes de permitir acesso aos dados. Caso não seja validado, uma exceção deve ser disparada;
- **RNF07**: Todos os métodos disponibilizados em EJB devem emitir exceções específicas quando houverem, a fim de que a aplicação GWT possa tratá-las conforme necessário.

3.4 DIAGRAMA ENTIDADE RELACIONAMENTO E DIAGRAMA DE CLASSES

Para que o sistema apresentado cumpra com seu objetivo, são necessárias as seguintes entidades:

- Categoria: contém dados das categorias e subcategorias que poderão ser utilizadas nas movimentações;
- Conta: armazena os dados das contas gerenciadas pelo usuário, às quais as movimentações são relacionadas;
- Movimento: contém os dados dos movimentos inseridos pelo usuário;
- Histórico: contém uma descrição e um tipo, que define entrada ou saída. Cada movimento deve ser relacionado com um histórico;
- Usuário: contém os dados dos usuarios que utilizam o sistema;
- Parâmetro: contém parâmetros para funcionamento do sistema, como servidor SMTP e sua porta de conexão, usuário e senha para envio do e-mail de confirmação entre outros dados.

Dessa maneira, o DER pode ser visualizado na Figura 9.

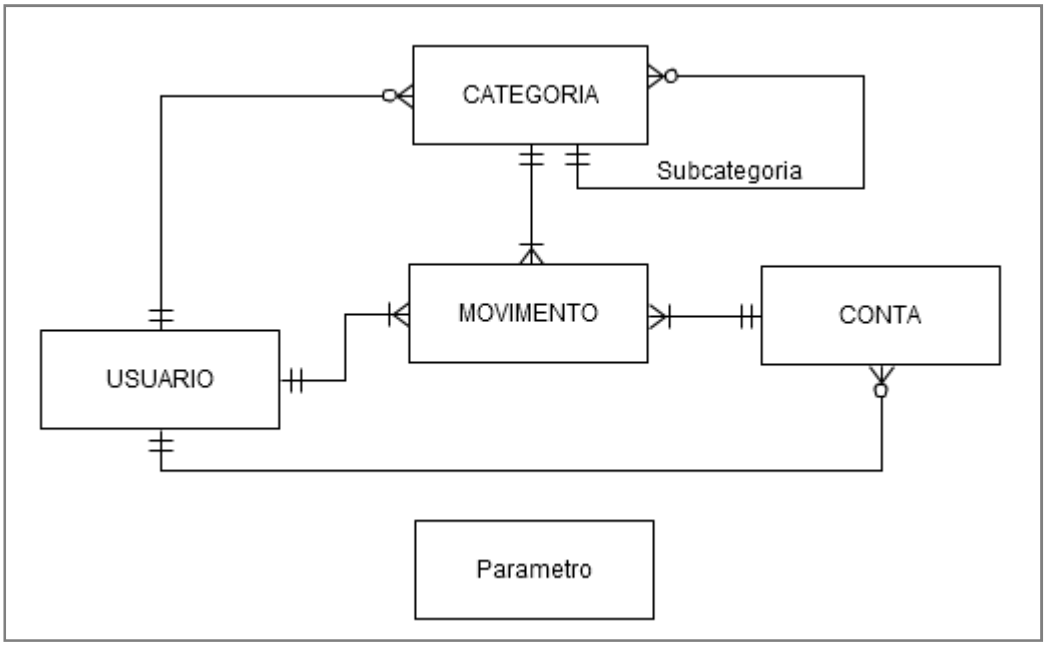

Figura 9 - DER do sistema

Na Figura 10, é exibido o diagrama de classes da aplicação.

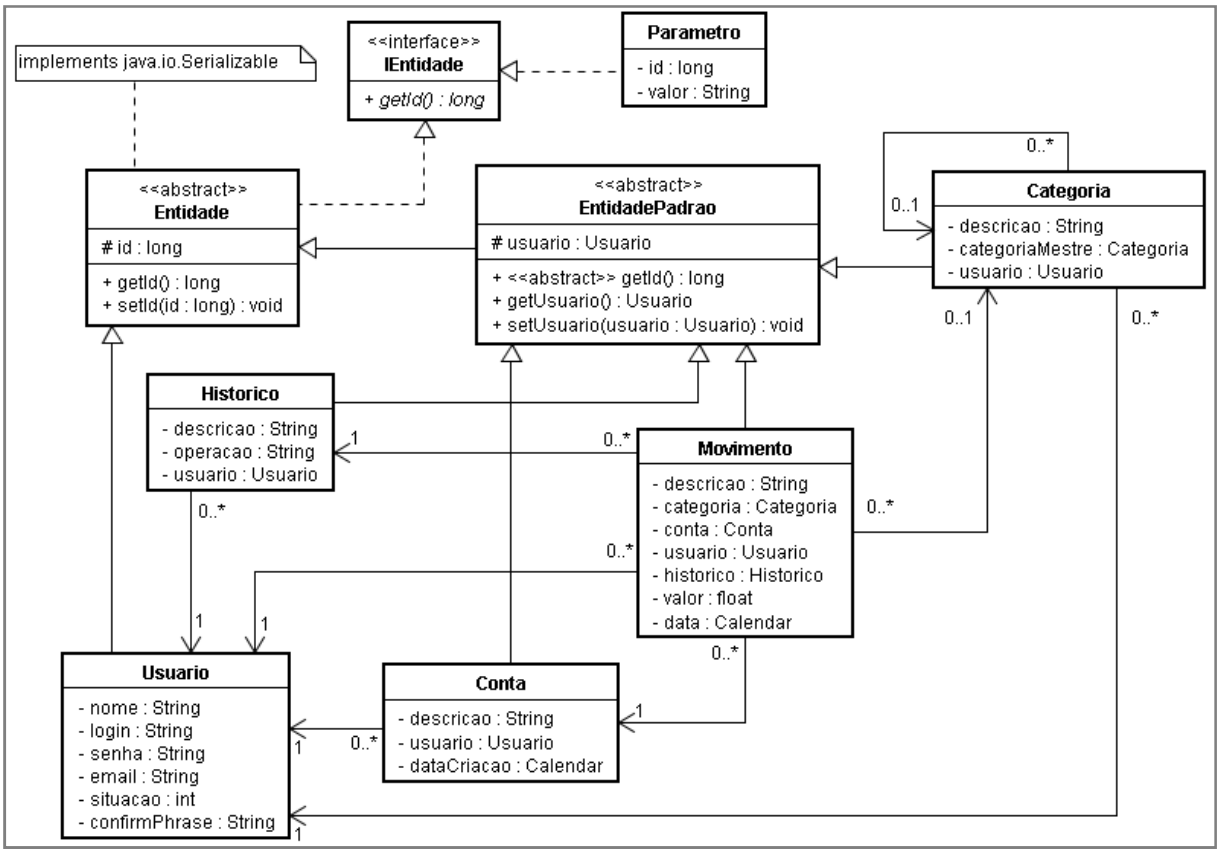

Figura 10 - Diagrama de classes da aplicação

3.5 CASOS DE USO

O sistema proposto possuirá apenas um ator: o AtorUsuário, que interage diretamente com o sistema.

Na Tabela 2 estão os casos de uso da aplicação a ser desenvolvida.

Tabela 2 - Casos de uso do sistema

Nº	Descrição	Entrada	Nome	Resposta
01	Usuário mantém movimentação	Dados movimentação, Dados categoria, Dados histórico, Dados conta	Manter Movimentação	msg01
02	Usuário mantém categoria	Dados categoria	Manter Categoria	msg02
03	Usuário mantém histórico	Dados histórico	Manter Histórico	msg03
04	Usuário mantém conta	Dados conta	Manter Conta	msg04
05	Usuário efetua *login*	Dados usuário	Efetuar Login	msg05
06	Usuário se cadastra	Dados usuário	Cadastrar Usuário	msg06
07	Usuário confirma e-mail	Usuário e código de confirmação	Confirmar E-mail	Msg07

O diagrama dos casos de uso pode ser visualizado na Figura 11.

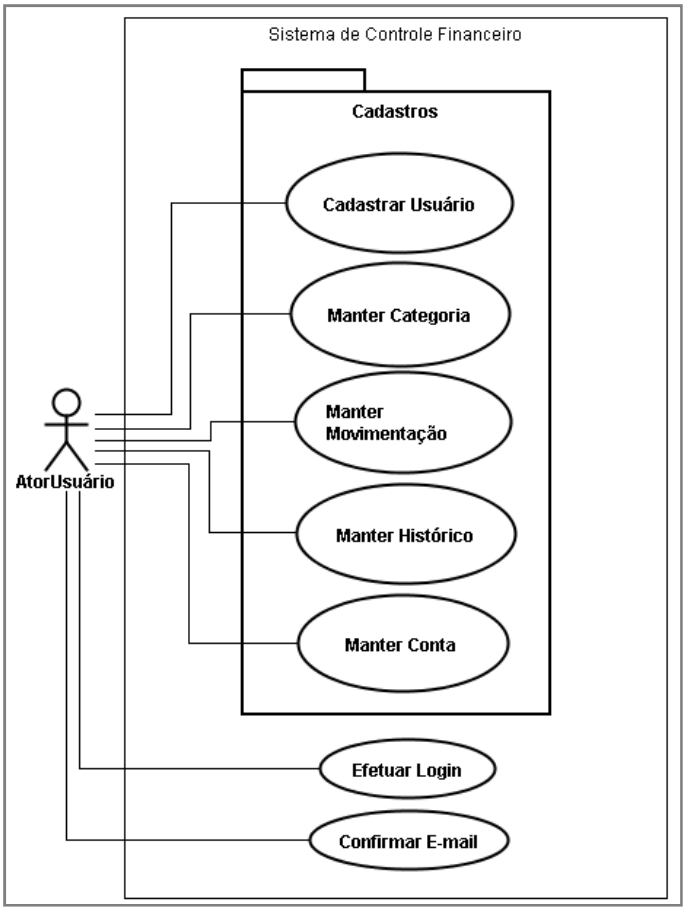

Figura 11 - Diagrama de casos de uso

3.5.1 Caso de Uso Efetuar Login

Caso de Uso	Efetuar Login - EL1
Atores	AtorUsuário (Iniciador)
Finalidade	Autenticar o usuário no sistema para que ele possa trabalhar com seus dados.
Visão Geral	O usuário pode efetuar o *login* no sistema informando sua senha, além de seu *login* ou e-mail.
Tipo	Secundária.
Pré-Condições	
Pós-Condições	O usuário efetuou *login* e pode trabalhar com seus dados.

Figura 12a - Caso de uso Efetuar Login

Ação do Ator	**Resposta do Sistema**
1. Informa senha, junto com *login* ou e-mail.	
2. Solicita *login*.	3. Valida o usuário.
	4. Apresenta uma lista das últimas movimentações do usuário, juntamente com painéis para alteração, consulta, inclusão ou exclusão de movimentos, categorias, contas e históricos.
Caminhos Alternativos	
2. Usuário desiste da operação. 2.1. Fim do caso de uso. 3. *Login*/E-mail ou senha não são válidos. 3.1. Retorna Msg05 com “Usuário ou senha inválidos”. 3.2. Retorna ao item 1. 3. Usuário ainda não confirmou seu e-mail. 3.1. Retorna Msg05 com “Você não confirmou seu e-mail ainda”. 3.2. Retorna ao item 1. 3. Usuário está desativado. 3.1. Retorna Msg05 com “Seu usuário está desativado”. 3.2. Retorna ao item 1.	

Figura 12b - Caso de uso Efetuar Login

3.5.2 Caso de Uso Manter Conta

Caso de Uso	Manter Conta – MC2
Atores	AtorUsuário (Iniciador)
Finalidade	Manter os dados cadastrais de uma conta.
Visão Geral	O usuário pode cadastrar, editar ou excluir uma conta do sistema.
Tipo	Secundária.
Pré-Condições	Usuário necessita ter efetuado *login* no sistema.
Pós-Condições	A conta foi salva e pode ser utilizada em uma movimentação ou foi excluída e não pode mais ser utilizada.

Figura 13a - Caso de uso Manter Conta

Ação do Ator	Resposta do Sistema
1. Opta por inserir nova conta.	
2. Informa os dados da conta.	
3. Confirma os dados.	4. Verifica se foram informados os campos obrigatórios.
	5. Verifica se já não existe conta cadastrada para este usuário com descrição igual à informada.
	6. Retorna Msg04 com “Conta salva”.

Caminhos Alternativos

1. Usuário opta por consultar uma conta para alteração ou exclusão.
  1.1. Usuário seleciona uma conta.
  1.2. Sistema permite edição ou exclusão da conta.
  1.3. Retorna ao item 2.

  1.1. Usuário desiste da operação.
    1.1.1. Fim do caso de uso.

2. Usuário exclui a conta.
  2.1. Sistema verifica se a conta não está sendo utilizado.
  2.2. Sistema retorna Msg04 com “Conta excluída”.
  2.3. Fim do caso de uso.

  2.1. Conta está sendo utilizado.
    2.1.1. Retorna Msg04 com “Exclusão não permitida. Conta em uso”.
    2.1.2. Retorna ao item 1.2.

2. Usuário desiste da operação.
  2.1. Fim do caso de uso.

3. Usuário desiste da operação.
  3.1. Fim do caso de uso.

3. Usuário exclui a conta.
  3.1. Retorna ao item 2.1.

4. Dados obrigatórios não informados.
  4.1. Informa ao usuário os campos que devem ser preenchidos.
  4.2. Retorna Msg04 com “Campos obrigatórios não informados”.
  4.3. Retorna ao item 2.

5. Já existe conta para este usuário com esta descrição.
  5.1. Retorna Msg04 com “Não é permitido inserir mais de uma conta com mesma descrição”.
  5.2. Retorna ao item 2.

Figura 13b - Caso de uso Manter Conta

3.5.3 Caso de Uso Cadastrar Usuário

Caso de Uso	Cadastrar Usuário – CU1
Atores	AtorUsuário (Iniciador)
Finalidade	Usuário se cadastra para utilizar o sistema.
Visão Geral	O usuário pode se cadastrar. Fazendo isso, ele receberá um e-mail de validação contendo um link para validar seu cadastro.
Tipo	Secundária.
Pré-Condições	
Pós-Condições	O está cadastrado no sistema e poderá confirmar seu cadastro a partir do e-mail de confirmação.

Ação do Ator	**Resposta do Sistema**
1. Informa dados do usuário.	
2. Solicita cadastro.	3. Verifica se todos os campos obrigatórios foram preenchidos.
	4. Solicita que o usuário informe código de confirmação (captcha).
5. Informa código de confirmação.	6. Valida código de confirmação.
	7. Valida dados do usuário.
	8. Envia e-mail de confirmação para o e-mail informado pelo usuário no cadastro.
	9. Retorna Msg06 com “Usuário cadastrado com sucesso! Um e-mail chegará em sua caixa para confirmar sua conta. Por favor, seja um pouco paciente”.

Caminhos Alternativos

2. Usuário desiste da operação.
  2.1. Fim do caso de uso.

3. Campos obrigatórios ausentes.
  3.1. Retorna Msg06 com “Campo obrigatório ausente”.
  3.2. Retorna ao item 1.

Figura 14a - Caso de uso Cadastrar Usuário

5. Usuário desiste da operação. 5.1. Fim do caso de uso. 6. Código de confirmação incorreto 6.1. Retorna Msg06 com “Código de confirmação incorreto”. 6.2. Retorna ao item 1. 7. E-mail já está em uso. 7.1. Retorna Msg06 com “E-mail já em uso”. 7.2. Retorna ao item 1. 7. *Login* já está em uso. 7.1. Retorna Msg06 com “Login já em uso”. 7.2. Retorna ao item 1.

Figura 14b - Caso de uso Cadastrar Usuário

3.5.4 Caso de Uso Manter Histórico

Caso de Uso	Manter Histórico - MH1
Atores	AtorUsuário (Iniciador)
Finalidade	Manter os dados cadastrais de um histórico.
Visão Geral	O usuário pode cadastrar, editar ou excluir um histórico do sistema.
Tipo	Secundária.
Pré-Condições	Usuário necessita ter efetuado *login* no sistema.
Pós-Condições	O histórico foi incluído ou alterado e pode ser utilizado em uma movimentação ou foi excluído e não pode mais ser utilizado.

Ação do Ator	**Resposta do Sistema**
1. Opta por inserir novo histórico.	
2. Informa os dados do histórico.	
3. Confirma os dados.	4. Verifica se foram informados os campos obrigatórios.
	5. Verifica se já não existe conta cadastrada para este usuário com descrição igual à informada.
	6. Retorna Msg03 com “Histórico salvo”.

Figura 15a - Caso de uso Manter Histórico

Caminhos Alternativos
1. Usuário opta por consultar um histórico para alteração ou exclusão. 1.1. Usuário seleciona um histórico. 1.2. Sistema permite edição ou exclusão do histórico. 1.3. Retorna ao item 2. 1.1. Usuário desiste da operação. 1.1.1. Fim do caso de uso. 2. Usuário exclui o histórico. 2.1. Sistema verifica se o histórico não está sendo utilizado. 2.2. Sistema retorna Msg03 com “Histórico excluído”. 2.3. Fim do caso de uso. 2.1. Histórico está sendo utilizado. 2.1.1. Retorna Msg03 com “Exclusão não permitida. Histórico em uso”. 2.1.2. Retorna ao item 1.2. 2. Usuário desiste da operação. 2.1. Fim do caso de uso. 3. Usuário desiste da operação. 3.1. Fim do caso de uso. 3. Usuário exclui o histórico. 3.1. Retorna ao item 2.1. 4. Dados obrigatórios não informados. 4.1. Informa ao usuário os campos que devem ser preenchidos. 4.2. Retorna Msg03 com “Campos obrigatórios não informados”. 4.3. Retorna ao item 2. 5. Já existe histórico para este usuário com esta descrição e tipo de histórico. 5.1. Retorna Msg03 com “Não é permitido inserir mais de um histórico com mesma descrição e tipo.”. 5.2. Retorna ao item 2.

Figura 15b - Caso de uso Manter Histórico

3.5.5 Caso de Uso Manter Categoria

Caso de Uso	Manter Categoria – MC1
Atores	AtorUsuário (Iniciador)
Finalidade	Manter os dados cadastrais de uma categoria.
Visão Geral	O usuário pode cadastrar, editar ou excluir uma categoria do sistema.
Tipo	Secundária.
Pré-Condições	Usuário necessita ter efetuado *login* no sistema.
Pós-Condições	A categoria foi salva e pode ser utilizada em uma movimentação ou foi excluída e não pode mais ser utilizada.

Ação do Ator	**Resposta do Sistema**
1. Opta por inserir nova categoria.	
2. Informa os dados da categoria.	
3. Confirma os dados.	4. Verifica se foram informados os campos obrigatórios.
	5. Verifica se já não existe categoria cadastrada para este usuário com descrição e a categoria mestre iguais às informadas.
	6. Retorna Msg02 com “Categoria salva”.

Caminhos Alternativos

1. Usuário opta por consultar uma categoria para alteração ou exclusão.
  1.1. Usuário seleciona uma categoria.
  1.2. Sistema permite edição ou exclusão da categoria.
  1.3. Retorna ao item 2.

  1.1. Usuário desiste da operação.
    1.1.1. Fim do caso de uso.

2. Usuário exclui a categoria.
  2.1. Sistema verifica se a categoria não está sendo utilizado.
  2.2. Sistema retorna Msg02 com “Categoria excluída”.
  2.3. Fim do caso de uso.

  2.1. Categoria está sendo utilizado.
    2.1.1. Retorna Msg02 com “Exclusão não permitida. Categoria em uso”.
    2.1.2. Retorna ao item 1.2.

Figura 16a - Caso de uso Manter Categoria

2. Usuário desiste da operação. 2.1. Fim do caso de uso. 3. Usuário desiste da operação. 3.1. Fim do caso de uso. 3. Usuário exclui a categoria. 3.1. Retorna ao item 2.1. 4. Dados obrigatórios não informados. 4.1. Informa ao usuário os campos que devem ser preenchidos. 4.2. Retorna Msg02 com “Campos obrigatórios não informados.”. 4.3. Retorna ao item 2. 5. Já existe categoria para este usuário com esta descrição. 5.1. Retorna Msg02 com “Não é permitido inserir mais de uma categoria com mesma descrição e categoria mestre”. 5.2. Retorna ao item 2.

Figura 16b - Caso de uso Manter Categoria

3.5.6 Caso de Uso Manter Movimentação

Caso de Uso	Manter Movimentação – MM1
Atores	AtorUsuário (Iniciador)
Finalidade	Manter os dados cadastrais de uma movimentação.
Visão Geral	O usuário pode cadastrar, editar ou excluir uma movimentação no sistema.
Tipo	Primária.
Pré-Condições	Usuário necessita ter efetuado *login* no sistema. A conta deve existir, o histórico deve existir, a categoria deve existir (se utilizada).
Pós-Condições	A movimentação foi salva ou foi excluída.

Ação do Ator	**Resposta do Sistema**
1. Opta por inserir nova movimentação.	
2. Informa os dados da movimentação.	
3. Confirma os dados.	4. Verifica se foram informados os campos obrigatórios.
	5. Retorna Msg01 com “Movimentação”.

Figura 17a - Caso de uso Manter Movimentação

Caminhos Alternativos
1. Usuário opta por consultar uma movimentação para alteração ou exclusão. 1.1. Usuário seleciona uma movimentação. 1.2. Sistema permite edição ou exclusão da movimentação. 1.3. Retorna ao item 2. 1.1. Usuário desiste da operação. 1.1.1. Fim do caso de uso. 2. Usuário exclui a movimentação. 2.1. Sistema exclui movimentação. 2.2. Sistema retorna Msg01 com “Movimentação excluída”. 2.3. Fim do caso de uso. 2. Usuário desiste da operação. 2.1. Fim do caso de uso. 3. Usuário desiste da operação. 3.1. Fim do caso de uso. 3. Usuário exclui a movimentação. 3.1. Retorna ao item 2.1. 4. Dados obrigatórios não informados. 4.1. Informa ao usuário os campos que devem ser preenchidos. 4.2. Retorna Msg01 com “Campos obrigatórios não informados”. 4.3. Retorna ao item 2.

Figura 17b - Caso de uso Manter Movimentação

3.5.7 Caso de Uso Confirmar E-mail

Caso de Uso	Confirmar E-mail - EL1
Atores	AtorUsuário (Iniciador)
Finalidade	Confirmar o e-mail do usuário para que, depois disso, ele possa se autenticar no sistema.
Visão Geral	Quando o usuário se cadastra, o sistema envia um e-mail de confirmação para o e-mail que ele informou com um código de confirmação. No e-mail enviado consta um *link* que, ao ser clicado pelo usuário, direciona-o para o sistema que, automaticamente, iniciará o processo de confirmação do e-mail com os parâmetros recebidos.

Figura 18a - Caso de uso Confirmar E-mail

Tipo	Secundária.
Pré-Condições	O usuário precisa ter se cadastrado no sistema e ter recebido o e-mail de confirmação.
Pós-Condições	O usuário está com o e-mail confirmado no sistema e pode efetuar *login*.

Ação do Ator	**Resposta do Sistema**
1. Abre o *link* recebido via e-mail parametrizado com seu usuário e código de confirmação.	2. Verifica se o usuário e o código de confirmação parametrizados são válidos.
	3. Altera a situação do usuário para usuário com e-mail confirmado.
	4. Retorna Msg07 com “E-mail confirmado com sucesso!”.

Caminhos Alternativos

2. Usuário informado é inválido.
  2.1. Retorna Msg07 com “Usuário inválido”.

2. Código de confirmação é inválido.
  2.1. Retorna Msg07 com “Frase de confirmação inválida”.

2. E-mail do usuário já foi confirmado.
  2.1. Retorna Msg07 com “Seu usuário já está ativado”.

2. Usuário foi desativado.
  2.1. Retorna Msg07 com “Seu usuário está desativado”.

Figura 18b - Caso de uso Confirmar E-mail

3.6 A ARQUITETURA DO SISTEMA

Conforme proposto, o sistema foi desenvolvido utilizando a tecnologia EJB para o lado servidor, disponibilizando os serviços, e a tecnologia GWT para o lado cliente, efetuando as devidas chamadas aos serviços do servidor. Para isso, a linguagem utilizada no desenvolvimento foi Java. O armazenamento dos dados é feito em um banco de dados MySQL, cujo acesso é feito apenas pelos serviços EJB.

Os seguintes EJBs são disponibilizados pelo servidor para serem acessados pelo GWT:

- `CategoriaBean`: responsável por cuidar da inserção, consulta, alteração e exclusão de categorias;
- `ContaBean`: responsável por cuidar da inserção, consulta, alteração e exclusão de contas;
- `HistoricoBean`: responsável por cuidar da inserção, consulta, alteração e exclusão de históricos;
- `MovimentoBean`: responsável por cuidar da inserção, consulta, alteração e exclusão de movimentos;
- `UsuarioBean`: responsável por cadastrar novos usuários, autenticá-los e validar o e-mail do usuário através do código de confirmação.

Já no GWT, os seguintes Servlets são disponibilizados para receber chamadas da camada do cliente e repassá-las para o EJB:

- `usuarioService`: responsável por receber pedidos de cadastro de novo usuário e de confirmação de e-mail do usuário;
- `categoriaService`: responsável por receber pedidos de inserção, alteração, exclusão e consulta de categorias;
- `contaService`: responsável por receber pedidos de inserção, alteração, exclusão e consulta de contas;
- `historicoService`: responsável por receber pedidos de inserção, alteração, exclusão e consulta de históricos;
- `loginService`: responsável por receber pedidos de *login* e *logout*;
- `movimentoService`: responsável por receber pedidos de inserção, alteração, exclusão e consulta de movimentos.

Os Servlets, através de chamadas utilizando JNDI, obtêm acesso aos serviços do EJB, podendo, então, repassar as chamadas recebidas do cliente e retornar para ele os resultados recebidos do servidor.

A Figura 19 apresenta o resultado final da tela inicial do sistema. Esta tela é exibida caso o usuário ainda não tenha efetuado *login*.

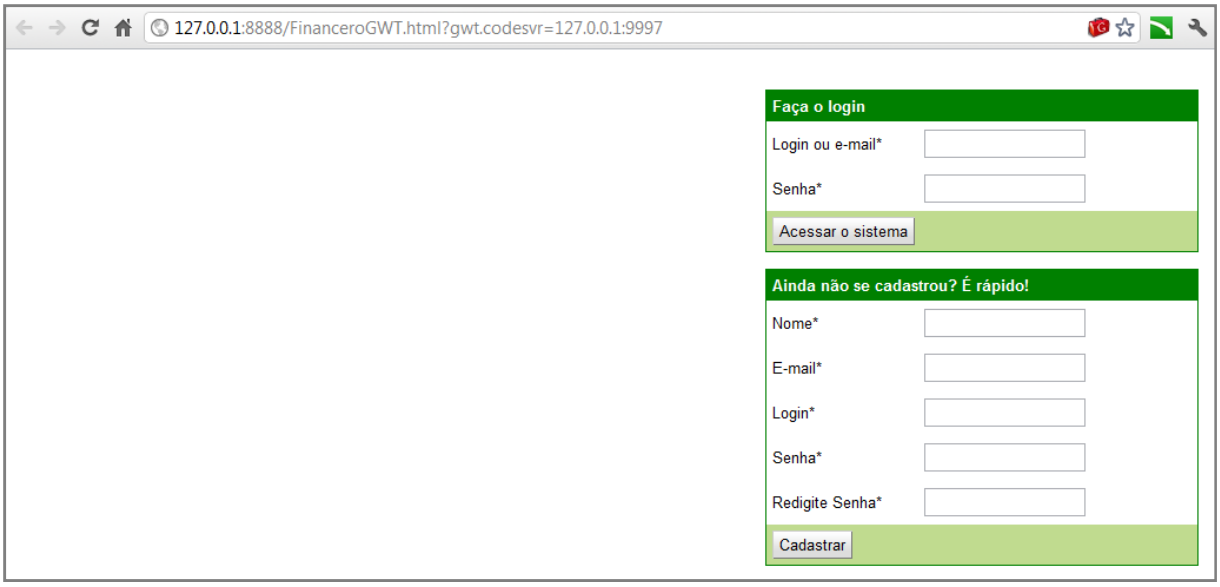

Figura 19 - Tela inicial do sistema (sem usuário autenticado)

3.7 TESTE DO SISTEMA

Um caso de teste é um conjunto de valores de entrada, pré-condições de execução, resultados esperados e pós-condições de execução desenvolvido para um objetivo particular ou uma condição de teste, como testar um caminho particular do programa ou a coerência dele com um requisito em específico (BLACK e MITCHELL, 2011).

Black e Mitchell (2011) definem uma condição de teste como um item ou evento de um sistema que pode ser verificado por um ou mais casos de teste como, por exemplo, uma função, uma transação, uma funcionalidade, um atributo de qualidade ou um elemento de estrutura.

O sistema proposto será testado e o planejamento de teste será documentado por um plano de teste. Um plano de teste é um documento descrevendo o escopo, a abordagem, os recursos e a programação das atividades de teste propostas (BLACK e MITCHELL, 2011). Identifica, entre outras coisas, as funcionalidades a serem testadas, as tarefas do teste, quem executará cada uma delas, o ambiente do teste e os critérios de entrada e saída que serão usados, por exemplo (BLACK e MITCHELL, 2011).

3.7.1 Plano de testes

O sistema de controle financeiro apresentará o seguinte plano de teste:

- Escopo: a aplicação a ser testada é Web e é visualizada através de um navegador. Ela funciona como um sistema de controle financeiro, aceitando lançamentos de valores em contas.
- Objetivos do teste: validar o funcionamento das principais funcionalidades propostas pelo sistema e se correspondem com os requisitos levantados.
- Resolução de defeitos: pelo motivo de a aplicação ser pequena, os defeitos, erros ou inconsistências serão corrigidos assim que encontrados.

Os procedimentos do teste, que são os casos de testes, serão apresentados em quadros. No presente trabalho o objetivo dos testes será testar apenas as principais funcionalidades do sistema de modo que conclua-se que a integração entre as tecnologias encontra-se em funcionamento.

No Quadro 20 e no Quadro 21 encontram-se os casos de teste que tratam de validar parte do caso de uso “Cadastrar Usuário”.

Teste 1.1	**Testar o cadastro de usuário com dados válidos**
Localização:	Página inicial do sistema.
Objeto de teste:	Verificar se encontra-se em funcionamento correto o auto cadastro de usuários no sistema.
Pré-condição:	1. Usuário não pode estar autenticado.
Procedimento:	1. Na página inicial, nos campos para cadastro de usuário, informar dados válidos; 2. Clicar em “Cadastrar”; 3. Informar corretamente o código de confirmação quando solicitado pelo sistema; 4. Clicar em “Confirmar”.

Resultado esperado:	1. O sistema deve exibir a mensagem “Usuário cadastrado com sucesso! Um e-mail chegará em sua caixa para confirmar sua conta. Por favor, seja um pouco paciente”; 2. O usuário deve receber um e-mail confirmando seu cadastro e contendo um *link* de confirmação de e-mail.

Quadro 20 - Caso de teste 1.1: cadastro de usuário com dados válidos

Teste 1.2	**Testar o cadastro de usuário sem informar campos obrigatórios**
Localização:	Página inicial do sistema.
Objeto de teste:	Verificar se o sistema aponta dados obrigatórios ausentes no cadastro de usuário.
Pré-condição:	1. Usuário não pode estar autenticado.
Procedimento:	1. Tentar submeter um formulário sem informar os campos obrigatórios: nome, e-mail, *login*, senha e confirmação de senha.
Resultado esperado:	1. O sistema deve apontar os campos obrigatórios que não foram informados.

Quadro 21 - Caso de teste 1.2: cadastro de usuário sem informar campos obrigatórios

O Quadro 22 exibe o caso de teste que objetiva verificar o funcionamento do sistema de acordo com o caso de uso “Confirmar E-mail”.

Teste 2.1	**Testar a confirmação de e-mail de usuário**
Localização:	Página inicial do sistema.
Objeto de teste:	Verificar se encontra-se em funcionamento correto a confirmação de e-mail do usuário.
Pré-condição:	1. O usuário precisa ter efetuado cadastro; 2. O usuário precisa ter recebido o e-mail com os dados para confirmação de seu endereço eletrônico.
Procedimento:	1. Abrir o *link* recebido via e-mail de confirmação.
Resultado esperado:	1. O sistema deve exibir a mensagem “E-mail confirmado com sucesso”.

Quadro 22 - Caso de teste 2.1: confirmação de e-mail de usuário

O Quadro 23 traz o caso de teste que validará se o sistema atende ao especificado no requisito funcional RF07 (usuário pode efetuar *login*). O Quadro 24, por sua vez, valida parte do caso de uso “Efetuar Login”, com relação ao caminho alternativo quando os dados informados não são válidos para autenticação do usuário.

Teste 3.1	**Testar o *login* do usuário com dados válidos**
Localização:	Página inicial do sistema.
Objeto de teste:	Verificar se encontra-se em funcionamento correto o *login* do usuário.
Pré-condição:	1. Usuário precisa estar cadastrado no sistema; 2. E-mail do usuário precisa ter sido confirmado; 3. Usuário não pode se encontrar na situação “desativado”.
Procedimento:	1. Informar um *login* e uma senha válidos; 2. Clicar em “Acessar o sistema”.
Resultado esperado:	1. O sistema não deve exibir mensagem alguma; 2. O sistema deve carregar a página inicial do usuário, permitindo que ele possa alterar seus dados.

Quadro 23 - Caso de teste 3.1: *login* do usuário com dados válidos

Teste 3.2	**Testar o *login* do usuário com dados inválidos**
Localização:	Página inicial do sistema.
Objeto de teste:	Verificar se o sistema acusa dados inválidos quando o *login* do usuário ou a senha não são informados corretamente.
Pré-condição:	1. Usuário não pode estar autenticado.
Procedimento:	1. Informar *login* e senha inválidos; 2. Clicar em “Acessar o sistema”.
Resultado esperado:	1. O sistema deve exibir a mensagem “Login ou senha inválidos”.

Quadro 24 - Caso de teste 3.2: *login* do usuário com dados inválidos

No Quadro 25 encontra-se o caso de teste que compreende parte do caso de uso “Manter Movimento”. Este caso de teste verificará o funcionamento da inclusão de novos movimentos no sistema.

Teste 4.1	**Testar a inserção de movimentos com dados válidos**
Localização:	Página inicial do sistema com o usuário autenticado.
Objeto de teste:	Verificar se encontra-se em funcionamento correto a inserção de novos movimentos de valores no sistema.
Pré-condição:	1. O usuário precisa estar autenticado no sistema; 2. A conta e o histórico usados na inserção devem estar previamente cadastrados.
Procedimento:	1. Informar dados válidos nos campos destinados à inclusão de movimentos.
Resultado esperado:	1. O sistema deve exibir a mensagem “Movimento salvo”; 2. O novo movimento deve estar disponível para edição ou exclusão.

Quadro 25 - Caso de teste 4.1: caso de uso inserção de movimento

4 RESULTADOS

Os testes foram realizados no seguinte ambiente:

- A aplicação EJB que trabalha do lado servidor foi rodada dentro de um servidor de aplicação Glassfish;
- A aplicação cliente desenvolvida em java utilizando GWT foi rodada em um servidor de aplicação Tomcat versão 6;
- Ambos os servidores Glassfish e Tomcat foram rodados na mesma máquina.

Os casos de teste foram executados e seus resultados serão apresentados em quadros. No Quadro 26, por exemplo, está o procedimento de teste executado para atender ao caso de teste 1.1. O Quadro 27 apresenta os dados utilizados neste teste.

Sequência	Passo	Resultado Esperado	Passou	Falhou
1	1. Dados informados; 2. Clicado no botão "Cadastrar".	1. Sistema solicita digitação de código de verificação (captcha).	X	
2	1. Código de confirmação digitado corretamente.	1. Sistema apresenta a mensagem "Usuário cadastrado com sucesso! Um e-mail chegará em sua caixa para confirmar sua conta. Por favor, seja um pouco paciente"; 2. Sistema envia um e-mail para o usuário com um link para validar seu e-mail.	X	

Quadro 26 – Resultado do caso de teste 1.1: cadastro de usuário com dados válidos

Campo	Valor
Nome	Nome Qualquer
E-mail	diegoselzlein@gmail.com
Login	diego
Senha	123
Redigite Senha	123

Quadro 27 - Dados utilizados no teste 1.1

A Figura 20 exibe a mensagem apresentada pelo sistema como resultado do caso de teste 1.1.

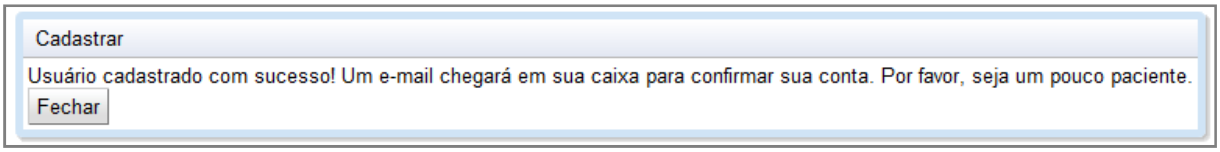

Figura 20 - Resposta do sistema para um cadastro de usuário efetuado com êxito

Os passos executados para o caso de teste 1.2 constam no Quadro 28.

Sequência	Passo	Resultado Esperado	Passou	Falhou
1	1. Nenhum dado informado; 2. Clicado no botão "Cadastrar".	1. Sistema indica campos obrigatórios que não foram informados.	X	

Quadro 28 - Resultado do caso de teste 1.2: cadastro de usuário sem informar campos obrigatórios

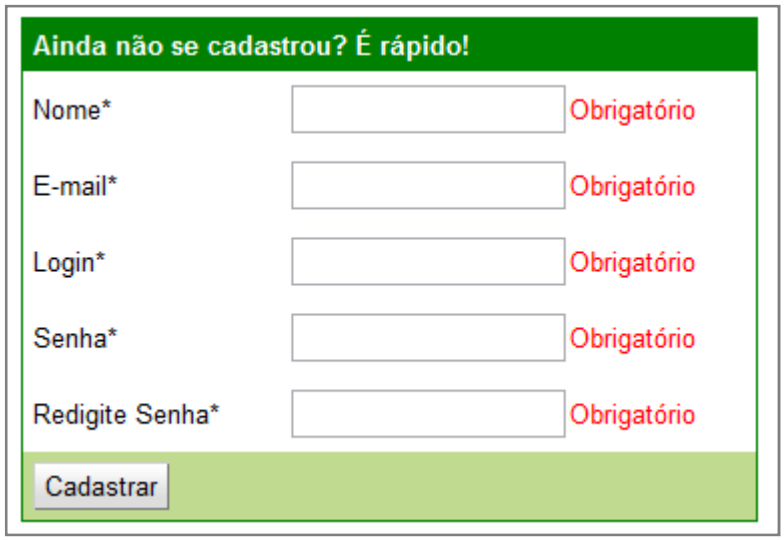

Figura 21 - Resultado do sistema para o caso de teste 1.2

O resultado do caso de teste 1.2 pode ser visualizado na Figura 21.

Após a realização do teste 1.1, o *link* de confirmação de e-mail recebido para aquele cadastro foi testado. O teste obteve êxito e o resultado encontra-se no Quadro 29.

Sequência	Passo	Resultado Esperado	Passou	Falhou
1	1. Aberto *link* de confirmação recebido via e-mail.	1. Sistema apresenta a mensagem “E-mail confirmado com sucesso”.	X	

Quadro 29 - Resultado do caso de teste 2.1: confirmação de e-mail de usuário

O Quadro 30 demonstra o resultado do teste 3.1, onde foi efetuado *login* do usuário cadastrado no teste 1.1. Os dados usados de entrada encontram-se no Quadro 31.

Sequência	Passo	Resultado Esperado	Passou	Falhou
1	1. Informados dados válidos; 2. Clicado no botão “Acessar o sistema”.	1. Sistema apresenta a tela inicial do usuário.	X	

Quadro 30 - Resultado do caso de teste 3.1: *login* do usuário com dados válidos

Campo	Valor
Login:	diego
Senha:	123

Quadro 31 - Dados utilizados no teste 3.1

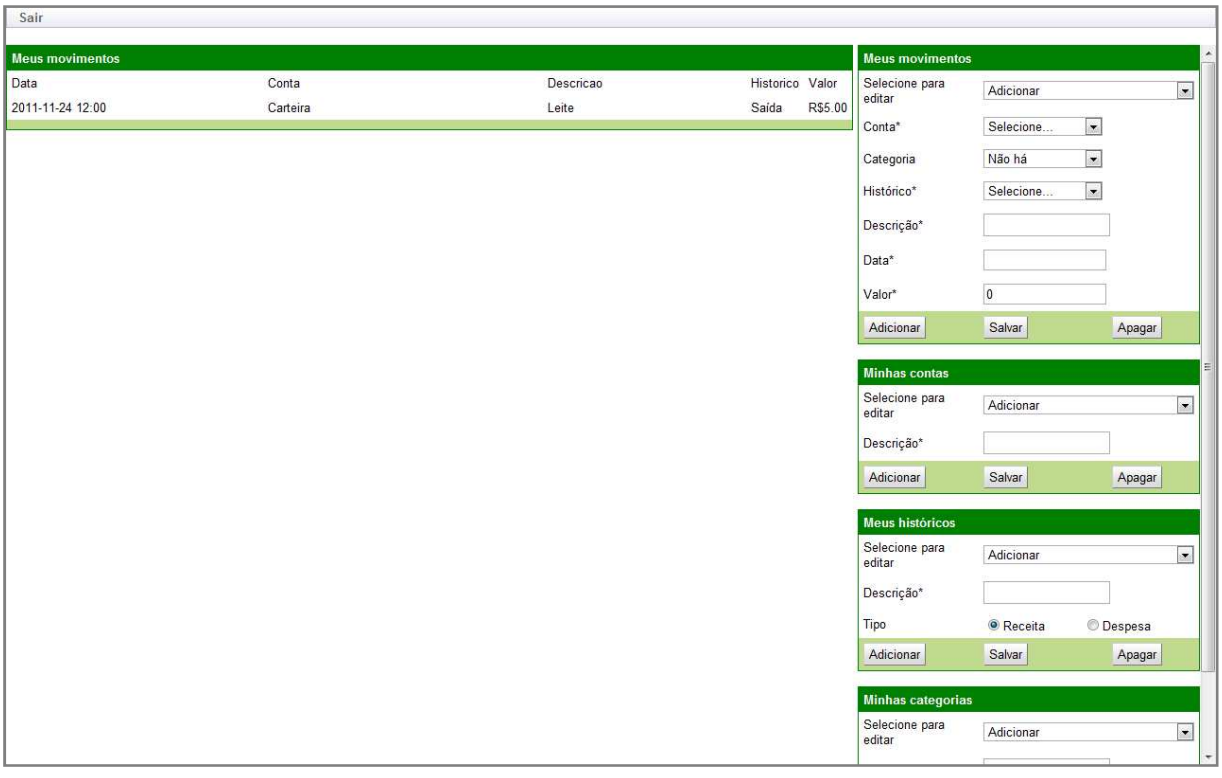

Figura 22 - Tela inicial do sistema com usuário autenticado

O resultado do caso de teste 3.1 pode ser visualizado na Figura 22, onde é exibida a tela inicial que o sistema apresenta para usuários que já efetuaram *login*.

O resultado obtido da execução do teste 3.2 se apresenta no Quadro 32 e os dados utilizados se encontram no Quadro 33.

Sequência	Passo	Resultado Esperado	Passou	Falhou
1	1. Informados dados inválidos; 2. Clicado no botão “Acessar o sistema”.	1. Sistema apresenta a mensagem “Usuário ou senha inválidos”.	X	

Quadro 32 - Resultado do caso de teste 3.2: *login* do usuário com dados inválidos

Campo	Valor
Login:	diego
Senha:	1234

Quadro 33 - Dados utilizados no teste 3.2

No Quadro 34 está o resultado obtido do teste 4.1, onde foi feita a inclusão de um movimento no sistema. Os dados informados para o movimento estão descritos no Quadro 35. Antes deste teste ter sido realizado, o usuário cadastrado no teste 1.1 foi autenticado, foi feita a inclusão de uma conta (a “Conta Corrente”) e de um histórico (com a descrição “Saída” e tipo “Saída”).

Sequência	Passo	Resultado Esperado	Passou	Falhou
1	1. Informados dados válidos; 2. Clicado no botão “Salvar”.	1. Sistema apresenta a mensagem “Movimento salvo”.	X	

Quadro 34 - Resultado do caso de teste 4.1: inserção de movimento

Campo	Valor
Conta:	Conta Corrente
Histórico:	Saída
Descrição:	Saque
Valor:	60

Quadro 35 - Dados utilizados no teste 4.1

5 CONSIDERAÇÕES FINAIS

Neste capítulo serão apresentadas as considerações finais e conclusões do presente trabalho. Além disso também serão listadas sugestões para possíveis trabalhos futuros.

5.1 CONCLUSÃO

Conforme apresentado, o *framework* EJB apresenta uma arquitetura robusta e diversas opções de configuração de semânticas para disponibilização de serviços, o que o torna uma ótima opção em aplicações cliente-servidor.

O segundo *framework* estudado, o GWT, possibilita a produção de sistemas Web, oferecendo uma arquitetura organizada e várias bibliotecas prontas para customização do sistema, como, por exemplo, internacionalização.

Separar dessa maneira um *software*, sendo uma aplicação EJB para servir e uma aplicação GWT para consumir permitiu a geração de uma aplicação com estrutura bem definida e dividida.

O *framework* GWT, pela forma que permite criar uma comunicação entre o JavaScript sendo executado no navegador e o Servlet disponibilizado por ele e pelo suporte à internacionalização do sistema, permitiu customizar o sistema.

Como pôde ser observado nos resultados dos testes, o sistema se comportou conforme desejado em suas funcionalidades mais importantes. Isto demonstra que a integração de tecnologias proposta obteve êxito e é uma boa opção para desenvolvedores que planejam desenvolver uma aplicação escalável, robusta e que apresente uma ótima experiência com o usuário.

5.2 TRABALHOS FUTUROS

Como sugestões para trabalhos futuros, os seguintes assuntos são sugeridos:

- Integração com Google Web Toolkit Platform (GWTP): o GWTP é uma arquitetura bem estabelecida moldada com o GWT, utilizando suas melhores ferramentas. Consiste em uma coleção de componentes que formam uma arquitetura. Uma sugestão para continuação do trabalho é a utilização dessa tecnologia juntamente com os *frameworks* GWT e EJB;
- Testes de desempenho: a aplicação desenvolvida foi testada apenas no quesito funcionamento. Porém, o trabalho apresentado não compreendeu testes de desempenho, tempo de resposta e eficiência, por exemplo, para definir se a integração de tecnologias proposta consiste em uma solução viável para uso em sistemas dirigidos ao mercado de *software*. Esta abordagem é uma sugestão de trabalho futuro;
- Boas práticas: de certa forma se relacionando com a sugestão sobre testes de desempenho, pode ser uma ideia de trabalho futuro uma pesquisa sobre boas práticas no desenvolvimento de aplicações EJB e GWT, com a finalidade de aprimorar o funcionamento do aplicativo e obter melhores resultados.

REFERÊNCIAS

ANDRETO, D. E.; AMARAL, A. M. M. M. **Engenharia de Software para Web**. Centro Universitário de Maringá. Maringá. 2006.

BLACK, R.; MITCHELL, J. L. **Advanced Software Testing**. Santa Barbara, CA: Rocky Nook, v. Vol. 3, 2011.

CHONOLES, M. J.; SCHARDT, J. A. **UML 2 For Dummies**. New York: Wiley Publishing, 2003.

FALBO, R. D. A. Engenharia de Software Notas de Aula, 2005. Disponivel em: <http://www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/NotasDeAula.pdf>. Acesso em: 3 ago. 2011.

GONCALVES, A. **Beginning Java EE 6 Platform with GlassFish 3:** From Novice to Professional. New York: [s.n.], 2009.

GUPTA, V. **Accelerated GWT:** Building Enterprise Google Web Toolkit Applications. [S.l.]: [s.n.], 2008.

HULL, E.; JACKSON, K.; DICK, J. **Requirements Engineering**. 3. ed. ed. Londres: Springer, 2011.

JUNIOR, C. G. B. Agregando Frameworks de Infra-Estrutura em uma Arquitetura Baseada em Componentes Um Estudo de Caso no Ambiente AulaNet, Março 2006. Disponivel em: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0410823_06_Indice.html>. Acesso em: 31 Agosto 2011.

LUZ, G. D. Enterprise Java Beans 3.1. **Java Magazine**, nov. 2009. Disponivel em: <http://www.devmedia.com.br/post-15013-Enterprise-Java-Beans-3-1.html>. Acesso em: 5 set. 2011.

ORACLE. Enterprise JavaBeans Technology, 200-. Disponivel em: <http://www.oracle.com/technetwork/java/javaee/ejb/index.html>. Acesso em: 5 set. 2011.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. ed. Rio de Janeiro, RJ: McGraw-Hill, 2006.

SMEETS, B.; URI, B.; BANKRAS, R. **Programando Google Web Toolkit:** do Iniciante ao Profissional. Rio de Janeiro: Alta Books, 2009.

SOMMERVILLE, I. **Engenharia de Software**. 6. ed. ed. São Paulo: Pearson Addison-Wesley, 2003.

SOMMERVILLE, I. **Software Engineering**. 8. ed. ed. [S.l.]: Pearson Education Limited, 2007.

SPINOLA, E. O. Introdução ao EJB 3.0 e o Enterprise Beans Components - Parte I. **DevMedia**, 2006. Disponivel em: <http://www.devmedia.com.br/post-1596-Introducao-ao-ejb-3-0-e-o-enterprise-beans-components-parte-i.html>. Acesso em: 5 set. 2011.

SRIGANESH, R. P.; BROSE, G.; SILVERMAN, M. **Mastering Enterprise JavaBeans 3.0**. Indianapolis: Wiley Publishing, 2006.

VOGEL, L. GWT Tutorial. **Vogella**, 2011. Disponivel em: <http://www.vogella.de/articles/GWT/article.html>. Acesso em: 3 out. 2011.

YOUNG, R. R. **The Requirements Engineering Handbook**. Norwood, MA: Artech House, 2004.